XML Web Services Security and Web based Application Security

XML Web Services are emerging as the fundamental building blocks for creating distributed integrated, interoperable solutions across the Internet. They represent a new paradigm in distributed computing that allow applications to be created from multiple XML Web Services dispersed across the web originating from various sources regardless of where they reside or how they were implemented. With all its promise of interoperability and ease of use, XML Web Services are severely hampered by the inherent lack of support for ...

XML Web Services Security and Web based Application Security

Chris Kwabi

GIAC Security Essentials Certification Practical Assignment Version 1.4b

Abstract

XML Web Services are emerging as the fundamental building blocks for creating distributed integrated, interoperable solutions across the Internet. They represent a new paradigm in distributed computing that allow applications to be created from multiple XML Web Services dispersed across the web originating from various sources regardless of where they reside or how they were implemented. With all its promise of interoperability and ease of use, XML Web Services are severely hampered by the inherent lack of support for security within initial versions of the standard. This threatens to limit widespread adoption of the technology, as concerns for confidentiality and message integrity across company boundaries are not adequately addressed. In recognition of this, efforts are currently underway to create a standardized security framework for XML Web services.

This paper provides high-level insights into how to create secure distributed, language neutral, platform independent web based applications using XML Web Services. A description of some of the efforts that are currently underway to create a standardized security framework for XML Web Services Security along with a representative example of a secure XML Web Services message.

Introduction

XML Web Services are emerging as the fundamental building blocks for creating distributed integrated, interoperable solutions across the Internet. They represent a new paradigm in distributed computing that allow applications to be created from multiple XML Web Services dispersed across the web originating from various sources regardless of where they reside or how they were implemented.

It provides an open standard, loosely coupled, language-neutral, platform-independent way of linking applications within an organization, across enterprises, and across the Internet. Unlike previous distributed computing technologies like CORBA and DCE, XML Web services are relatively simple and easy to implement and stand to transform the web from the standard click and pull model that is prevalent today to a distributed workflow model that connects the underpinnings of commerce together across the globe. Businesses will be able to communicate with relative ease in ways that were previously not feasible.

With all its promise of interoperability and ease of use, XML Web Services are severely hampered by the inherent lack of support for security within initial versions of the standard. This threatens to limit widespread adoption of the technology, as concerns for confidentiality and message integrity across company boundaries are not adequately addressed within the initial standard protocols. In recognition of this, efforts are currently underway to create a standardized security framework for XML Web services. The recently released WS-Security standard, which defines the hooks to ensure end-to-end message integrity, confidentiality and security of web services, provides the foundation for this framework.

This paper attempts to highlight the current state of affairs and discusses efforts that are currently underway to rectify the situation. Today, because of the ubiquity of the HTTP protocol across the web, XML Web Services applications are generally built on the same infrastructure as traditional web applications. We start, therefore, by presenting some of the most common security vulnerabilities frequently found in traditional web applications and provide recommendations on how to avoid them. Next we provide a description of the XML Web Services Security framework that is currently being developed and end by providing a more in depth look at the pillar of the of the security framework, the WS-Security specification.

XML Web Services Overview

Many formal definitions of “XML Web Services” exist. However, there are several basic characteristics that they all possess.

- XML Web Services operate over standard protocols and technologies like XML, HTTP, TCP/IP, SMTP and others. Today the de facto transport mechanism for XML Web Services is HTTP protocol. The message

protocol, however, is defined to operate over any number of different transport mechanisms.

- XML Web Services expose functionality to web users through a standardized protocol called SOAP and is the de facto communication standard for exchanging XML Web Services messages. It is a specification that defines an XML format for message exchanges between communicating parties involved in a web services session. It defines a simple extensible XML messaging format that can be used over multiple protocols with a variety of differing programming models (e.g. requst/response, RPC). It also defines a complete processing model that outlines how messages are processed as they traverse through a path and provides a rich flexible framework for defining higher-level application protocols that offer increased flexibility and interoperability in distributed heterogeneous environments. Because of the ubiquity of HTTP on the web today, the initial SOAP specification includes definitions for how SOAP is used with HTTP and RPC. Invocations of other transport mechanisms for SOAP messages are currently not standardized and consequently may cause interoperability issues.
- XML Web Services provide a standard mechanism to describe their interfaces that allow potential users to build solutions that incorporate functionality provided by the XML Web Service. This standard service description mechanism is provided in the form of an XML document format called the Web Services Description Language (WSDL).
- XML Web Services can be registered so that other potential users can find them using another component of the standard called Universal Discovery Description Interface (UDDI)

In securing web services each of these components described need to be considered. In this document, however, we focus on securing the messaging component and play particular attention to securing SOAP messages.

XML Web Services Security Introduction

One of the biggest concerns about the potential use of web services is the concern of vulnerabilities within the infrastructure that may allow malicious users to attack their system. These attacks could result in limiting availability, private data being compromised, or loosing control of a machine. Furthermore, Web Services based systems face some of the same security issues that currently afflict traditional web applications. When an organization puts up a web application, they invite the whole world to send them HTTP requests. Attacks buried in these requests sail past firewalls, filters, platform hardening and intrusion detection systems without notice because they are inside legal HTTP requests. Even secure websites that use SSL typically just accept the requests that arrive through the encrypted tunnel without scrutiny. Some basic guidelines for constructing secure web based applications along with common web

application security vulnerabilities are presented below. These guidelines are applicable to XML Web Services based applications as well.

Common Web Application Security Vulnerabilities

In crafting secure XML web services based solutions it is important to be aware of the most common web application security vulnerabilities that are typically exploited in traditional web based applications. Because of the ubiquity of HTTP in the web today, XML Web Services are generally deployed over HTTP. Consequently these security vulnerabilities are also applicable to the majority of XML Web Services applications as well. A summary of the most significant web application security vulnerabilities is presented below. Web based application developers need to be aware of the susceptibilities in the infrastructure so they can design systems that are immune to these exploits.

Uncorroborated parameters

When web applications neglect to validate information included in web requests before using this data, they render themselves susceptible to parameter tampering based attacks. Web applications (XML Web Services included) use HTTP requests to determine what actions to take. Attackers can tamper with any part of an incoming HTTP request including the URL, query string, headers, cookies, form fields, and hidden fields to bypass the web site's security infrastructure. Some of the common parameter tampering attacks include forced browsing, command insertion, SQL injection, cookie poisoning and hidden field manipulation. Unfortunately, a significant number of web applications are designed to use client side input validation only which can easily bypassed, leaving web applications exposed without any real protection against malicious parameters. It is relatively simple for an attacker to generate their own HTTP requests and sidestep any of the security mechanisms included within the client. Therefore, it is important to implement server side checks to defend against parameter manipulation attacks. In fact, a huge number of attacks on web applications would become extremely difficult or nearly impossible if server side validation was included in web-based designs.

The best defense against these types of attacks is to ensure that all parameters are validated before they used. Each parameter should be validated against exact rules that specify what input will be allowed. Guidelines on how parameter validation should be conducted include checking for the following;

- Data type (string, integer, real, etc)
- Allowable character set
- Minimum and maximum length
- Whether a null is allowed or not
- Whether the parameter is required or not
- Whether duplicates are allowed
- Numeric ranges
- enumerated values
- specific patterns

Broken Access Control

In the context of building secure web application, access control refers to how web applications go about granting access to content and functionality to some users while denying it to others. Access control mechanisms in web applications are typically tied to the content and functionality that the web site provides. When restrictions to constrain authenticated users to only perform activities that they authorized to are not properly implemented, attackers can exploit this defect to access other user's accounts, view sensitive data, or use unauthorized functions. Web application access control mechanism's are generally not well thought out and typically morph over time without a consistent global approach. Access control mechanisms in web applications should be based on the associated security policy and be included in the security architecture from the beginning of the design cycle. The security policy should specify what types of users can access the system and what types of actions and content each class of users are allowed to access.

Broken Account and Session Management

Each web application must establish sessions to keep track of the stream of requests directed at the application from each user. HTTP does not inherently provide session management so web applications must generally handle this themselves. Today, most of the common web application environments provide this capability themselves. However, if session tokens are not adequately protected, attackers can easily take over an active session and assume the identity of the original user. Creating strong session tokens and ensuring that they are sufficiently protected throughout their active lifecycle is an essential element of a secure web application. Another major vulnerability included in this category is associated with backend authentication. Backend authentication refers to how web applications authenticate themselves to backend systems such as databases, directories, and web services. Web applications must have access to the appropriate security credentials in order to access these systems. This implies that if the web server is compromised these backend systems may be at risk. Often plaintext passwords and other easily exploitable security credentials are directly embedded in source code and configuration files. Provisions should be made to ensure that security credentials needed to access backend systems from within the web application are sufficiently protected.

Cross-Site Scripting (XSS) Flaws

Cross site scripting (XSS) refers to an exploit that can transpire when an attacker uses a web site to send malicious code, usually written in a scripting language like JavaScript, to another user. When a web site sends information out that it receives from one user to another user without checking the data then that web site is vulnerable to cross site-scripting attacks. A malicious user can insert harmful code in the input and use the web site to forward it other users. These

attacks exploit the trust that these targeted end users have in the web site to perform tasks that would not be normally allowed. These messages are often disguised by using things like Unicode to make them look less suspicious. Users are typically tricked into clicking on a link or submitting a form, which causes the injected code to travel to the vulnerable web server. The web server then reflects the attack back to the unsuspecting user's browser, which then executes the code because it appears to come form a trusted server.

XSS attacks are categorized into two types depending on how the malicious payload is introduced into the server. Stored XSS attacks generally refer to attacks where the malicious code is permanently stored on the server in for example a database, message forum etc. Reflected XSS attacks are introduced via alternate mechanisms like email or other servers.

Like many other attacks, the best defense against XSS attacks is to perform rigorous code reviews of all parameters to make sure that only permitted data are allowed by these parameters.

Buffer Overflows

Buffer overflows are mechanism used by intruders to corrupt the execution stack of web applications or for that matter any other application that is vulnerable to this type of an attack. By sending carefully crafted input messages to a web application, an attacker can cause the web application to execute arbitrary code that can be designed to take control of the machine. They are generally difficult to detect and extremely difficult to exploit. Buffer overflows can be found in the web server products, application server products and the web applications themselves.

Keeping up with the latest bug reports and software updates for your web and application server can significantly reduce the likelihood of these attacks. For custom applications, ensuring that all code that accepts inputs from HTTP requests perform the appropriate size and type checking on these inputs.

Command Injection Flaws

Web applications typically pass parameters when they need to access external systems or the local OS. If an attacker can embed malicious commands in these message parameters, the external system can then be made to execute these commands on behalf of the web application. These attacks can include calls to the OS via system calls, calls to external programs via shell commands, and calls to backend databases using SQL commands (for e.g. SQL injection). Similarly, scripts written in Perl, Python and other scripting languages can be inserted into poorly designed web applications and executed. Web applications that use interpreters are particularly vulnerable to this type of an attack.

A large number of web applications employ OS features and external programs to provide functionality within the application. When a web application passes

along data from an HTTP request to an external system, it must carefully check this information before passing it along to verify that this data is confined to what is allowed. If this is not done, attackers can inject special characters, malicious code, or command modifiers into the data and these poorly designed web application will blindly pass along this data for execution by the external system.

Error handling Problems

The improper handling of errors in the code of a web application can result in a variety of security vulnerabilities for the web site. In poorly designed systems, detailed internal error messages such as stack traces, database dumps and are displayed to the user. These messages divulge system implementation details that can be exploited by a knowledgeable hacker.

Web applications, like most other applications, generate error conditions during the course of normal operation. Examples of these errors include out of memory, null pointer exceptions, system call failures, database unavailable, network timeout and a host of others. Proper handling of these errors within the application should provide meaningful error messages to the user, diagnostic information to administrators and no useful information about the internal workings of the web site to potential intruders.

Good error handling design implementations should be capable of handling any feasible set of inputs and still enforce the requisite security. Straightforward error messages should be generated and logged so that their cause can be reviewed later to determine whether they were triggered by an error in the web site or by an attacker. Error handling should not only focus on user inputs, but should include errors generated by internal components such as system calls, database queries, or any other internal calls.

Insecure Use of Cryptography

Most web applications today are required to store sensitive information within the application infrastructure either in a database or in a file system somewhere. Sensitive data like passwords, credit card numbers, account data and other proprietary data are often stored within the application using encryption. Unfortunately, developers frequently overestimate the protection provided by encryption and are careless in securing other aspects of the infrastructure. Common areas where mistakes are made include:

- Insecure storage of encryption keys, certificates and passwords
- Improper storage of secrets in memory
- Poor random generators
- Poor encryption algorithm choice
- Failure to encrypt sensitive data
- Inventing new encryption algorithms
- Failure to include support for security credential maintenance procedures, like key changes

The best way to avoid cryptographic errors is to minimize the use of cryptography unless absolutely necessary. When it must be used, cryptographic libraries that have been substantially exposed to public scrutiny should be used. Furthermore, cryptographic functions should be encapsulated and carefully reviewed.

Remote Administration Flaws

Administrative interfaces to web sites provide powerful features for managing a web application that can be exploited by intruders if not adequately protected. The interfaces typically allow administrators to efficiently manage users, data, and content on their site. Not surprisingly however, these interfaces are frequently the focus of attacks by intruders both inside and outside the infrastructure due to their enormous power.

Common problems include weak authentication and/or encryption on these administrative web interfaces, lack of enforcement of keeping lesser administrators restricted to areas where they are authorized, flaws in the separation between users and administrators, and making unnecessarily powerful administrative functions available.

Web and Application Server Misconfiguration

The configurations of the web and applications servers used by a web site play a critical role in ensuring the security of the web site. These servers are responsible for serving up content and invoking applications that generate content. Application servers may also provide other useful functions to the web application, like data storage, directory services, mail, messaging etc. Some of the sever configuration problems that can plague a site include:

- Unpatched security flaws in the server software
- Server flaws and configurations that allow directory listing and directory traversal attacks
- Unnecessary default, backup or sample files, including scripts, applications, configuration files and web pages
- Improper file and directory permissions
- Unnecessary services enabled including content management and remote administration
- Default accounts with their default passwords
- Administrative or debugging functions that are enabled or accessible
- Overly informative error messages
- Improperly configured SSL certificates and encryption settings
- Use of self-signed certificates to achieve authentication and man-in-the-middle protection
- Use of default certifications

Intruders can use any number of security scanning tools available on the market to detect many of these flaws and once detected, use these vulnerabilities to exploit the web site. Ensuring that both the web and applications servers have

been hardened and updated with the latest security patches is an essential first step in preventing this kind of an exploit.

Implications for XML Web Services based applications

Avoiding these pitfalls and following the recommendations provided above to create secure web based application infrastructures that are not susceptible to any of the vulnerabilities mentioned above is an essential first step in ensuring that the web application is securely protected.

XML Web Services based applications present an interesting problem in this light, however. Users of a poorly designed XML Web services site may inadvertently expose themselves to security vulnerabilities by interfacing with this site and vice versa. It is essential therefore to validate that the Web Services systems that you interface to, do not inadvertently expose your application to malicious exploits.

XML Web Services Security Framework

The benefits of using an open standard, loosely coupled, platform independent way of linking applications within organizations, across the enterprise and over the Web is becoming apparent as more and more XML Web Services pilots and production systems are being deployed. By far, the most significant promise of the emerging XML Web Services paradigm is the ability to deliver, integrated interoperable solutions that span organizational boundaries. However, the lack of a comprehensive security model within the existing architecture to ensure the integrity, confidentiality and security of the Web Services architecture is a serious draw back to the wide spread adoption of this technology. Before wide spread adoption can truly be expected from the industry, a comprehensive security framework must be developed for the technology. In recognition of this deficiency, a number of industry proponents (IBM, Microsoft etc), have joined together to put together a plan and roadmap for developing a set of specifications that address how these messages can be protected in a Web services environment. This effort attempts to create a security model that brings together previously incompatible security technologies like public key infrastructure and Kerberos and is being designed to facilitate the creation of secure Web services in the heterogeneous IT world that is characteristic of environments where this is expected to be utilized. A broad set of specifications are being developed that cover security technologies like authentication, authorization, privacy, trust, integrity confidentiality, secure communications channels, federation, delegation and auditing across a wide spectrum of application and business topologies. This section describes the vision of this effort and provides an indication of where they are in the process and what needs to be done.

By leveraging the built in extensibility inherent in the XML Web Services model, the specification builds upon foundational technologies like SOAP, WSDL, XML digital Signatures, XML Encryption and SSL/TLS. The WS-Security specification

represents the corner stone of this effort and defines the core facilities for protecting the integrity and confidentiality of messages within this protocol. It also provides mechanisms for associating security related claims with the message. WS-Security specification is only the beginning of a much larger effort to produce specifications that deal with policy, trust and privacy issues.

Web Services Security Model Motivation

Providing a comprehensive security framework for web services requires integration of currently available processes and technologies with the evolving security requirements of the emerging web services applications. It requires solutions to both technological (secure messaging) and business processes (policy, trust, risk), and requires coordinated efforts by vendors, application developers, network and infrastructure providers and users of these services.

To achieve this, the functional requirements of application security must be abstracted from the particular security mechanisms used. For example, a user performing an online transaction should not be impacted by whether he uses a cell phone or a desktop computer as long as each device can securely provide the proper identity. The goal of this security framework is to allow the creation of interoperable solutions with relative ease using heterogeneous systems. For example, the secure messaging model supports both PKI and Kerberos identity mechanisms and views them as particular incarnations of a broader facility that is capable of being extended to support other identity mechanisms. Integration accomplished within the framework of a single security model abstraction enables entities to still use their existing investments in security technology and yet be able to securely communicate with other entities that use different security technologies. Furthermore, as organizations with differing identity mechanisms attempt to collaborate using web services, the security trust model provides a flexible framework that can be used to allow them to interoperate when configured with the appropriate authorization.

Each customer and every web service has their own set of security requirements based on their particular business needs and operational environment. Since these requirements typically vary and are often expressed in different levels of specificity, a successful approach to the web services security model requires a set of flexible interoperable security primitives that through the use of policy and configuration enable a variety of secure solutions

To address these challenges, efforts are underway in industry and in the standards bodies that propose an evolutionary approach to creating secure, interoperable Web services based on a set of security abstractions that unify formerly dissimilar technologies. As an example of this evolutionary approach, the secure messaging model can be added to existing transport level security. This approach for example, could add message level integrity and or persistent confidentiality to an existing web service whose messages are currently carried

through a Secure Sockets Layer tunnel. A more detailed description of this evolutionary web services security framework is provided below.

Web Services Security Model Terminology

In order to avoid any ambiguity, a number of the terms that are used to describe the proposed security framework are described below.

Web Service

The term "Web service" can apply to a wide variety of network base application topology. In this context, the term "Web Service" describes application components whose functionality and interfaces are exposed to potential users through the use of existing and emerging Web technology standards including XML, SOAP, WSDL, and HTTP. In contrast to web sites, browser based interactions or platform dependent solutions, Web services are computer-to-computer based services that are generated via defined formats and protocols in a platform independent and language neutral manner.

Security Token

Security tokens are defined as representations of security related information. Examples include x.509 certificates, Kerberos tickets, mobile device security tokens from SIM cards, and usernames.

Signed Security Token

Signed security tokens are security tokens that contain a set of related claims that have been cryptographically endorsed by an issuer. Examples include X.509 certificates and Kerberos tickets.

Claims

A claim is a statement about a subject that is generated either by the subject or by relying party that associates the subject with the statement. No restrictions are placed on the types of claims that can be made or on how these claims are expressed. Claims cab be used for example to assert the sender's identity or authorized role.

Subject

The subject of a security token is the entity (person, application or business) about which the claims expressed in the security token apply. The subject of a security token possesses information that is necessary to prove ownership of the security token.

Proof-of-Possession

This refers to the information used in the process of proving ownership of a security token or set of claims. The private key associated with the public key contained in a security token would be considered proof-of-possession.

Web Service End Policy

Web services have complete flexibility in specifying the claims they require in order to process requests. Collectively, these required claims and related information is referred to as the Web Service End Policy. End policies can be expressed in XML and can be used to specify requirements related to authentication, authorization, or other custom requirements.

Claim Requirements

Claim requirements can be applied to entire messages or parts of a message, to all actions of a particular type or actions under a given set of conditions. An example of this would be a service requiring a requestor to prove authorization to purchase above a certain amount.

Intermediaries

As SOAP messages traverse through a system from the initial requestor to a service end point, they may be operated on by intermediaries that perform actions such as message routing, or even message modification. Examples include intermediaries adding headers to the message, encrypting or decrypting portions of the message or adding security tokens. Care should however be taken to ensure that message alterations do not destroy message integrity, violate trust models or destroy accountability.

Actor

An actor is an intermediary or end point that is identified by a URI and that processes SOAP messages.

Web Service Security Model Principles

XML Web Services are generally accessed by sending SOAP messages that request specific actions from the targeted service end points. In return, these end points process these requests and send back SOAP message responses to the requesting parties. Within the context of the web services security framework, securing XML web services essentially entails securing the integrity and confidentiality of these messages and ensuring that the targeted service only acts on requests that provide the necessary claims requested by the operating policies associated with the service.
Today, the de facto standard for providing transport level security for web services is the secure sockets layer (SSL)/transport layer security (TLS) protocol. SSL/TLS typically enable point-to-point secure sessions that provide security features like authentication, data integrity and confidentiality. SOAP messages are typically sent using HTTP over SSL/TLS, which implies that, each node along a SOAP message's route will be exposed to potentially sensitive data that may be contained in the message.

IPSec is a network layer security protocol that can also be used to secure XML web services applications. In scenarios where the communication end points and intermediaries are known before hand, secure IPSec sessions can be set up a

priori to provide the necessary network level security. Like SSL, IPSec provides similar security features like host authentication, data integrity and confidentiality, although at a much lower level in the OSI stack and can be configured to secure point-to-point sessions between communicating parties in a web services set up.

Both these security mechanisms rely on protocols that are defined outside of the SOAP message protocol itself. Although SOAP is typically associated with HTTP, the SOAP message format is not tied to a single protocol. It is conceivable for a SOAP message to be routed over multiple protocols as it travels towards its destination

Web services applications exist in a wide variety of configurations that include mobile devices, gateways, proxies, load-balancers, demilitarized zones (DMZs), outsourced data centers, and globally distributed dynamically configured systems. These systems rely on the ability of the message processing intermediaries to forward messages from one intermediary to another until they reach their final destination. The SOAP messaging model operates on logical end points that abstract the physical network and the application infrastructure and consequently frequently incorporate a multi-hop topology that may include any number of intermediary actors. A direct consequence of this is that the confidentiality and integrity of data in the messages flowing through an intermediary may be compromised after transport level processing is completed between two intermediary. This compels upstream message intermediaries to rely on security evaluations made by previous intermediaries and to completely trust their handling of the messages. This may not be acceptable in certain scenarios where highly sensitive information is transported within these messages and calls for a comprehensive end-to-end Web Service Security architecture to secure communications between two logical end points involved in a session. This comprehensive security architecture should leverage both transport and application level security mechanisms to achieve this.

The proposed security framework includes the following characteristics:

- A Web service can require an incoming message to prove a set of claims (e.g. name, key, permission, etc) that it may require. The service may reject or ignore messages that arrive without the required claims. The set of required claims and related information is referred to as the policy.
- A requestor can send messages with proof of the required claims by associating security tokens with the messages. These messages demand a specific action as well as prove that their sender has the right to demand the action.
- When a requestor does not have the necessary claims, the requestor or someone on its behalf can attempt to retrieve the necessary credentials by contacting other web services. These other services are referred to as Security Token Services and may require their own set of claims. Security token services broker trust between different rust domains by issuing security tokens

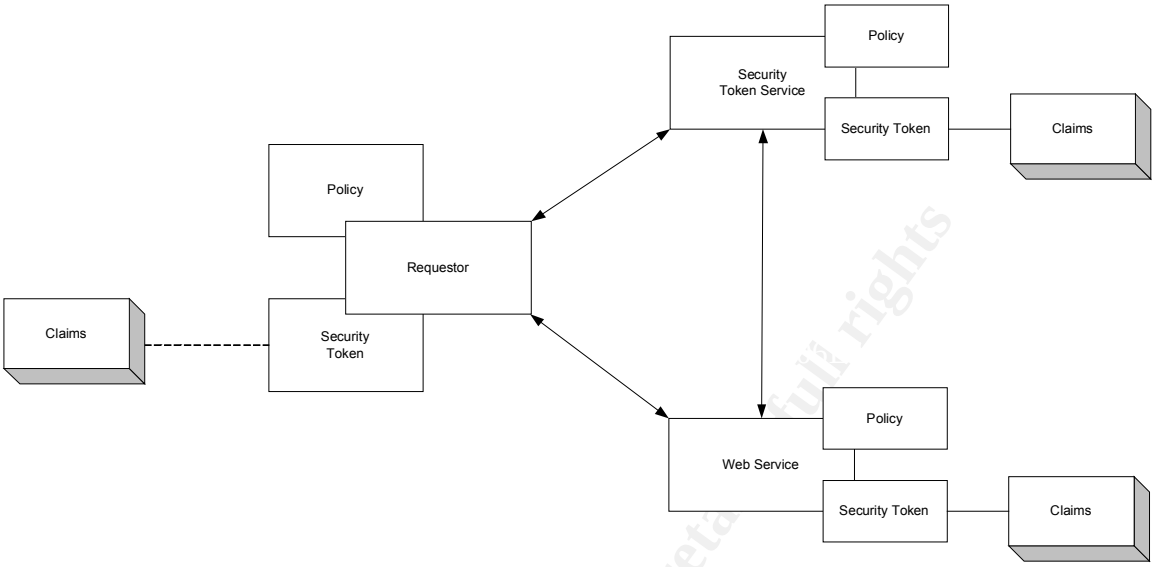

Figure 1: Generalized XML Web Services Security Framework (Source: "Security in a Web Services World: A Proposed Architecture and Roadmap")

This generalized secure messaging framework depicted above is based on claims, policies, and tokens can support several specific security models like identity-based-security, access control lists, and capabilities based security. It allows the use of existing technologies such as x.509 public key certificates, Kerberos shared secret tickets, and even password digests. The framework provides an integrating abstraction that allows systems to bridge diffident security technologies and is sufficient to construct higher-level key exchange, authentication, authorization, auditing and trust mechanisms.

Web Services Security Framework Specifications

The strategy described above and the WS-Security specifications described below provide the framework and foundation for the proposed Web services security model. The set of security specifications that will be included in this model is presented below and will be based on the message security specification (WS-Security)

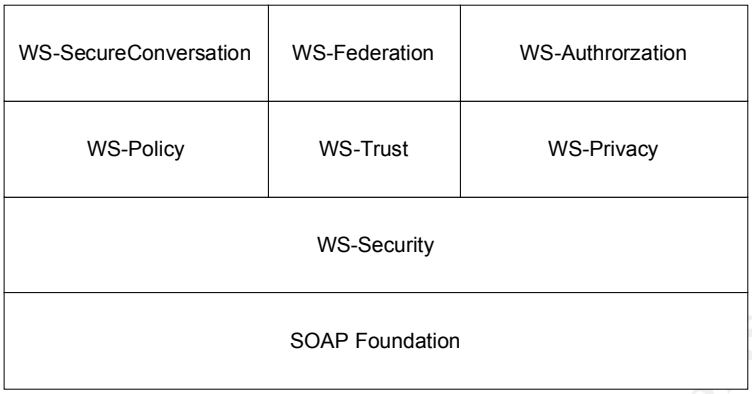

Figure 2: XML Web Services Security Framework Specifications (Source: "Security in a Web Services World: A Proposed Architecture and Roadmap")

Initial Specifications

- WS-Security: This specification which is the cornerstone of the framework (currently in existence) describes how to attach digital signatures and encryption headers to SOAP messages. It also describes how to attach security tokens, including binary encoded tokens such as X.509 certificates and Kerberos tickets, to these messages.
- WS-Policy: This specification, which is currently undefined, will describe the capabilities and constraints of the security (and business) policies on intermediaries and endpoints (e.g. required security tokens, supported encryption algorithms, privacy rules.)
- WS-Trust: this specification (currently under development) will define a framework for trust models that enables web services to securely interoperate.
- WS-Privacy: This specification (currently under development) will describe a model for how Web services and requestors stat privacy preferences and organizational privacy practice statements

Follow on Specifications

- WS-SecureConversation: This specification will describe how to manage and authenticate message exchanges between entities including security context and establishing and deriving session keys.
- WS-Federation: This specification will describe how to manage and broker trust relationships within a heterogeneous federated environment including support for federated entities.
- WS-Authorization: This specification will describe how to manage authorization data and authorization policies.

WS-Security: SOAP Message Security

The WS-Security specification describes enhancements to the SOAP messaging specification to include message integrity and message confidentiality. It also defines how to attach and include security tokens within SOAP messages. Finally, it includes a facility for specifying binary encoded security tokens like X.509 certificates. Each of these enhancements can be used independently or in conjunction with each another to accommodate a wide range of security models.

WS-Security provides a general-purpose mechanism for associating security tokens with messages. No particular type of security token is required and it is designed to support multiple security token formats.

Message integrity is provided by leveraging XML Signature in conjunction with security tokens (which may contain or imply key data) to ensure that messages are transmitted without modifications. The integrity mechanisms are designed to support multiple signatures, potentially by multiple actors and to be extensible to add additional signature formats.

Similarly, it leverages the XML Encrytion standard in conjunction with security tokens to keep portions of SOAP messages confidential thus providing message confidentiality. It includes support for additional encryption technologies, processes, and operations by multiple actors. The encryption may reference a security token.

Finally, WS-Security includes a mechanism for encoding binary security tokens. It describes how to encode X.509 certificates, Kerberos tickets and also includes opaque encrypted keys. It also includes extensibility mechanism that can be used to further describe the characteristics of the security token included with the message.

Extended Example

A representative example of a secure SOAP message using the WS-Security specification is presented below. The example is provided without delving into the specific details of each of the constructs defined within the WS-Security specification and is provided for illustrative purposes. The example is fairly involved and for the uninitiated may appear intimidating. The message demonstrates the use of timestamps, digital signatures, X.509v3 token key, symmetric key encryption XML Web Services message constructs within the example.

```
(001)    <?xml version="1.0" encoding="utf-8"?>
(002)     <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
               xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
               xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext
          "
               xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/xx/utility"
```

```
                    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
(003)          <S:Header>
(004)                <wsu:Timestamp>
(005)                     <wsu:Created wsu:Id="T0">
(006)                          2001-09-13T08:42:00Z
(007)                     </wsu:Created>
(008)                </wsu:Timestamp>
(009)                <wsse:Security>
(010)                     <wsse:BinarySecurityToken
                               ValueType="wsse:X509v3"
                               wsu:Id="X509Token"
                               EncodingType="wsse:Base64Binary">
(011)                          MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
(012)                     </wsse:BinarySecurityToken>
(013)                     <xenc:EncryptedKey>
(014)                          <xenc:EncryptionMethod Algorithm=
                                    "http://www.w3.org/2001/04/xmlenc#rsa-
                               1_5"/>
(015)                          <wsse:KeyIdentifier
EncodingType="wsse:Base64Binary"
(016)                               ValueType="wsse:X509v3">MIGfMa0GCSq...
(017)                          </wsse:KeyIdentifier>
(018)                     <xenc:CipherData>
(019)
<xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
(020)                          </xenc:CipherValue>
(021)                     </xenc:CipherData>
(022)                     <xenc:ReferenceList>
(023)                          <xenc:DataReference URI="#enc1"/>
(024)                     </xenc:ReferenceList>
(025)                </xenc:EncryptedKey>
(026)                <ds:Signature>
(027)                     <ds:SignedInfo>
(028)                          <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(029)                          <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
(039)                          <ds:Reference URI="#T0">
(031)                               <ds:Transforms>
(032)                                    <ds:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(033)                               </ds:Transforms>
 (034)                          <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
(035)                          <ds:DigestValue>LyLsF094hPi4wPU...
(036)                          </ds:DigestValue>
```

```
(037)                                   </ds:Reference>
(038)                                       <ds:Reference URI="#body">
(039)                                        <ds:Transforms>
(040)                                            <ds:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(041)                                        </ds:Transforms>
(042)                                       <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
(043)
      <ds:DigestValue>LyLsF094hPi4wPU... 1
(044)                                       </ds:DigestValue>
(045)                                   </ds:Reference>
(046)                                </ds:SignedInfo>
(047)                               <ds:SignatureValue>
(048)                               Hp1ZkmFZ/2kQLXDJbchm5gK...
(049)                          </ds:SignatureValue>
(050)                          <ds:KeyInfo>
(051)                               <wsse:SecurityTokenReference>
(052)                                   <wsse:Reference URI="#X509Token"/>
(053)                               </wsse:SecurityTokenReference>
(054)                          </ds:KeyInfo>
(055)                    </ds:Signature>
(056)              </wsse:Security>
(057)               </S:Header>
(058)                    <S:Body wsu:Id="body">
(059)                    <xenc:EncryptedData
Type="http://www.w3.org/2001/04/xmlenc#Element" wsu:Id="enc1">
(060)                          <xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#3des-cbc"/>
(061)                          <xenc:CipherData>
(062)
      <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
(063)                               </xenc:CipherValue>
(064)                          </xenc:CipherData>
(065)                    </xenc:EncryptedData>
(066)               </S:Body>
(067)   </S:Envelope>
```

Figure 3. Secure SOAP Message Example – source; " Web Service Security: SOAP Message Security"

A review some of the key sections of the example is provided below. Readers are encouraged to refer to the "WSS SOAP Message Secuirty-11" specification for more background.

Lines (003)-(057) contain the SOAP message headers.

Lines (004)-(008) specify the timestamp information. In this case it indicates the creation time of the message.
Lines (009)-(056) represent the <wsse:Security> header block. This contains the security-
related information for the message.
Lines (010)-(012) specify a security token that is associated with the message. In this case, it specifies an X.509 certificate that is encoded as Base64. Line (011) specifies the actual Base64 encoding of the certificate.
Lines (013)-(025) specify the key that is used to encrypt the body of the message. Since this is a symmetric key, it is passed in an encrypted form.
Line (014) defines the algorithm used to encrypt the key.
Lines (015)-(017) specify the name of the key that was used to encrypt the symmetric key.
Lines (018)-(021) specify the actual encrypted form of the symmetric key.
Lines (022)-(024) identify the encryption block in the message that uses this symmetric key. In this case it is only used to encrypt the body (Id="enc1").
Lines (026)-(055) specify the digital signature. In this example, the signature is based on the X.509 certificate.
Lines (027)-(046) indicate what is being signed. Specifically, Line (039) references the creation timestamp and line (038) references the message body.
Lines (047)-(049) indicate the actual signature value – specified in Line (042).
Lines (051)-(053) indicate the key that was used for the signature. In this case, it is the X.509 certificate included in the message. Line (052) provides a URI link to the Lines (010)-(012). The body of the message is represented by Lines (056)-(066).
Lines (059)-(065) represent the encrypted metadata and form of the body using XML Encryption.
Line (059) indicates that the "element value" is being replaced and identifies this encryption. Line
(060) specifies the encryption algorithm – Triple- DES in this case. Lines (062)-(063) contain the actual cipher text (i.e., the result of the encryption). Note that we don't include a reference to the key as the key references this encryption – Line (023).

Source; " Web Service Security: SOAP Message Security"

Security Considerations

As was pointed out earlier, the WS-Security specification provides mechanisms for ensuring message integrity and confidentiality as well as the ability to include associated security tokens within the message structure. It does not, however, include support for establishing trust between communicating parties. For example, when an application relies on digital signatures to establish trust, security components outside the scope of the WS-Security specification like certificate validation and processing are required to ensure the validity of cryptographic keys that are used for the basis of the trust. Existing specifications

like the XML Key Management Specification (XKMS) and future specifications like WS-Trust mentioned above can be used to provide Trust services between communicating parties. Key management life cycle issues associated with administration of cryptographic keys are necessary to establish the basis of the security infrastructure. Some of the related issues include

- Secure generation of keys,
- Secure distribution of keys,
- Secure revocation of keys,
- Controlling the use of the different type of keys such that they are only used for their intended purpose,
- Managing access to keys, which is a matter of ensuring that personnel and applications only use keys they have been authorized to use,
- Keeping an audit trail of key usage.

Applications using the WS-Security specification are required to establish and manage validity of the cryptographic keys before including them within these secure message constructs.

It is also strongly recommended that SOAP messages exchanged over an open network like the Internet include digitally signed elements to ensure their integrity. However, since imposters can record signed messages and send them back as replay attacks, it is important to combine the use of digital signatures with other security mechanisms like timestamps, sequence numbers or nonces to ensure the uniqueness of the message and to prevent replay attacks.

Digital signatures used for the verification of claims by the sender should be confirmed by requiring the sender to prove knowledge of the corresponding private key. This can be established through a challenge response mechanism whereby the relying party sends a message that the sender is required to sign and subsequently send back for verification. Once again, timestamps, sequence numbers or expirations should be used to ensure the integrity of these messages and to prevent replay attacks.

Security tokens appearing in <wsse:Security> header elements must be signed by their issuing authority to provide message recipients with the assurance that the messages have not been forged or altered since issuance. Furthermore, message senders should be sure to sign any security token elements that it can voucher for that have not been signed by their issuing authority.

The incorrect application of the combination of signing and encrypting a common data element can introduce security vulnerabilities. For example, if digitally signed data left in the clear, is accompanied by the encrypted digitally signed data counterpart, this would leave the system vulnerable to plain text guessing attacks.

Message Ids and timestamps should also be signed to ensure that they have not been modified in any way. Signed timestamps can be used to counter replay attacks by caching the most recent timestamp from a particular service for a given period of time. A minimum of five minutes should be used as a guideline for detecting replay attacks for timestamps and nonces in highly interactive sessions.

Passwords included in these messages should be adequately protected if the underlying transport mechanisms do not provide adequate protection against eavesdropping.

Conclusion

As more and more companies evolve their IT infrastructures to include XML Web Services as the primary mechanism for interoperating with business partners and the like, it will be essential for them to understand the security implications of using XML Web Service message constructs within their web based applications. The efforts currently underway to create a standardized security framework to facilitate interoperability and end-to-end security amongst heterogeneous systems involved in XML Web Service communications sessions will go a long way in promoting wide spread adoption of this open standard web based message protocol.

References

1. OASIS. "Web Services Security: SOAP Message Security" Working Draft 11. URL http://www.oasis-open.org/committees/documents.php (3 March 2003).
2. Open Web Security Application Project (OWASP)."The Ten Most Critical Web Application Security Vulnerabilities". URL http://www.owasp.org/ (13 January, 2003)
3. Matt Powell, Microsoft Corporation. "Real SOAP Security" URL http://msdn.microsoft.com/webservices/building/default.aspx?pull=/library/en-us/dnservice/html/service11212001.asp (21 November, 2001)
4. Open Web Security Application Project (OWASP). "A Guide to Building Secure Web Applications". URL http://www.owasp.org/ (11 September, 2002)
5. WC3. "XML Key Management Specification (XKMS)". URL http://www.w3.org/TR/xkms (30 March, 2001)
6. Jeannine Hall Gailey, Microsoft Corporation, "Encrypting SOAP Messages Using Web Services Enhancements". URL http://msdn.microsoft.com/webservices/building/default.aspx?pull=/library/en-us/dnwebsrv/html/wseencryption.asp (March, 2003)
7. Joint whitepaper by IBM and Microsoft. "Security in a Web Services World: A Proposed Architecture and Roadmap". URL

http://msdn.microsoft.com/webservices/understanding/default.aspx?pull=/library/en-us/dnglobspec/html/ws-rm-exec-summary.asp (March 13, 2003)
8. Roger Wolter, Microsoft Corporation, “XML Web Services Basics”, URL http://msdn.microsoft.com/webservices/understanding/default.aspx?pull=/library/en-us/dnwebsrv/html/webservbasics.asp (December 2001)
9. Aaron Skonnard, Microsoft Corporation. “Understanding XML Schema”. URL http://msdn.microsoft.com/webservices/understanding/default.aspx?pull=/library/en-us/dnxml/html/understandxsd.asp (March 2003).

Last Updated: July 20th, 2018

Upcoming SANS Training

Click here to view a list of all SANS Courses

SANS Riyadh July 2018	Riyadh, SA	Jul 28, 2018 - Aug 02, 2018	Live Event
SANS Pittsburgh 2018	Pittsburgh, PAUS	Jul 30, 2018 - Aug 04, 2018	Live Event
Security Operations Summit & Training 2018	New Orleans, LAUS	Jul 30, 2018 - Aug 06, 2018	Live Event
SANS Hyderabad 2018	Hyderabad, IN	Aug 06, 2018 - Aug 11, 2018	Live Event
Security Awareness Summit & Training 2018	Charleston, SCUS	Aug 06, 2018 - Aug 15, 2018	Live Event
SANS Boston Summer 2018	Boston, MAUS	Aug 06, 2018 - Aug 11, 2018	Live Event
SANS San Antonio 2018	San Antonio, TXUS	Aug 06, 2018 - Aug 11, 2018	Live Event
SANS August Sydney 2018	Sydney, AU	Aug 06, 2018 - Aug 25, 2018	Live Event
SANS New York City Summer 2018	New York City, NYUS	Aug 13, 2018 - Aug 18, 2018	Live Event
SANS Northern Virginia- Alexandria 2018	Alexandria, VAUS	Aug 13, 2018 - Aug 18, 2018	Live Event
SANS Krakow 2018	Krakow, PL	Aug 20, 2018 - Aug 25, 2018	Live Event
Data Breach Summit & Training 2018	New York City, NYUS	Aug 20, 2018 - Aug 27, 2018	Live Event
SANS Chicago 2018	Chicago, ILUS	Aug 20, 2018 - Aug 25, 2018	Live Event
SANS Prague 2018	Prague, CZ	Aug 20, 2018 - Aug 25, 2018	Live Event
SANS Virginia Beach 2018	Virginia Beach, VAUS	Aug 20, 2018 - Aug 31, 2018	Live Event
SANS San Francisco Summer 2018	San Francisco, CAUS	Aug 26, 2018 - Aug 31, 2018	Live Event
SANS Copenhagen August 2018	Copenhagen, DK	Aug 27, 2018 - Sep 01, 2018	Live Event
SANS SEC504 @ Bangalore 2018	Bangalore, IN	Aug 27, 2018 - Sep 01, 2018	Live Event
SANS Wellington 2018	Wellington, NZ	Sep 03, 2018 - Sep 08, 2018	Live Event
SANS Amsterdam September 2018	Amsterdam, NL	Sep 03, 2018 - Sep 08, 2018	Live Event
SANS Tokyo Autumn 2018	Tokyo, JP	Sep 03, 2018 - Sep 15, 2018	Live Event
SANS Tampa-Clearwater 2018	Tampa, FLUS	Sep 04, 2018 - Sep 09, 2018	Live Event
SANS MGT516 Beta One 2018	Arlington, VAUS	Sep 04, 2018 - Sep 08, 2018	Live Event
Threat Hunting & Incident Response Summit & Training 2018	New Orleans, LAUS	Sep 06, 2018 - Sep 13, 2018	Live Event
SANS Baltimore Fall 2018	Baltimore, MDUS	Sep 08, 2018 - Sep 15, 2018	Live Event
SANS Alaska Summit & Training 2018	Anchorage, AKUS	Sep 10, 2018 - Sep 15, 2018	Live Event
SANS Munich September 2018	Munich, DE	Sep 16, 2018 - Sep 22, 2018	Live Event
SANS London September 2018	London, GB	Sep 17, 2018 - Sep 22, 2018	Live Event
SANS Network Security 2018	Las Vegas, NVUS	Sep 23, 2018 - Sep 30, 2018	Live Event
SANS DFIR Prague Summit & Training 2018	Prague, CZ	Oct 01, 2018 - Oct 07, 2018	Live Event
Oil & Gas Cybersecurity Summit & Training 2018	Houston, TXUS	Oct 01, 2018 - Oct 06, 2018	Live Event
SANS Brussels October 2018	Brussels, BE	Oct 08, 2018 - Oct 13, 2018	Live Event
SANS Pen Test Berlin 2018	OnlineDE	Jul 23, 2018 - Jul 28, 2018	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced