A Security Architecture for Web Services

Hikmat Farhat, Khalil Challita
Computer Science Department
Notre Dame University-Louaize
Lebanon
{hfarhat,kchallita}@ndu.edu.lb

ABSTRACT: *Web services are quickly becoming the most popular tool for distributed computing. Due to this popularity a comprehensive security architecture is needed. In this paper we introduced such a comprehensive architecture that includes- in addition to the standard services of integrity and confidentiality- authentication, authorization and a defense against denial of service attacks. This model builds on existing standards such as SOAP, WSS and XACML. A detailed implementation with performance evaluation using the open source Apache tools is also discussed.*

Keywords: Web Services, Security, WSS, XACML, SOAP, Denial of Service

Received: 1 March 2011, Revised 8 April 2011, Accepted 13 April 2011

1. Introduction

While the use of the term Web Services is ubiquitous there is no consensus on the meaning of the term. The World Wide Web Consortium defines web services as "a software application identified by a URI" and "whose interfaces and bindings are capable of being defined, described, and discovered". This is the definition that we will use throughout this paper. Furthermore web services are transforming the web from a data oriented repository to a service oriented repository [7]. The importance of web services stems from the fact that in many settings the conventional middle approach does not work or at least it is unwieldy. Most Business to Business interaction these days are done either by filling forms on the Internet or through email. Such interactions while using technology are still done manually because in multi-organizational interactions there is no obvious way where to put the middleware. Also for every pair or group of ssss businesses a different middleware might be needed. Since many companies have more than one partner this leads to the situation where a given company has to support many middleware systems, one for each partner.

With the added flexibility of Web Services comes the added complexity of securing these services. Therefore a comprehensive model for securing web services is a must. Dealing with the security of Web Services is not a new issue but it lacks, to date, a comprehensive architecture where all components are integrate in a single model.

Our aim in this paper is to provide a combined security model for web services that ensures both authentication and authorization, in addition to the standard services of integrity and confidentiality. This is essential for web services in order to allow a client and a service to communicate securely, and be protected from potential problems such as unidentified client requests or unauthorized access to resources. Our model combines components that belong to different perspectives of security technologies such as *Web Service Security (WS Security)* standards, application level protocol, and application layer processing.The rest of this paper is divided as follows. The main concepts in Web Services are introduced in Section 2. The main issues in securing web

services are discussed in 3. The overall architecture of our model is presented in 4. The related work is presented in Section 5. In Section 6 we give our implementation our model that ensures integrity, confidentiality, authentication and authorization as well as protection against DoS. Concluding remarks are given in Section 7.

2. Web Services Architecture

Before discussing issues relating to the security of web services it is important to describe the entity we are trying to secure, namely the web service architecture. Toward that end it is important to divide the architecture of web service into two parts: internal and external. While the security of the internal architecture is important which includes viruses sandbox randomized space we are mostly concerned with the external architecture of web services and toward this end we provide authentication, authorization and protection from denial of service attacks using puzzles.

2.1 SOAP

The Simple Object Access Protocol underlies all interactions between Web Services [7]. SOAP is standardized by the W3C, specifically by the XML Protocol Working Group, which is part of the Web Services Activity [1].

SOAP organizes information to be exchanged between participants in SOAP messages and it specifies how a client can invoke a remote procedure using a SOAP message. The processing model used by the standard assumes a SOAP message originates at an initial SOAP sender and is sent to an ultimate SOAP receiver via zero or more SOAP intermediaries. SOAP messages are made of an envelope which contains an optional header and a mandatory SOAP body. The body can optionally contain a subpart for error reporting. In this paper the SOAP Header is the most important part of the message because SOAP headers are typically used to transmit authenticaion or session management data. SOAP can be transported either by SMTP or HTTP. While HTTP is more popular, SMTP is typically used for asynchronous communication. Another advantage of HTTP is that it can handle firewalls and by using HTTPS one can immediately get some security. In fact this is the recommendation of Web Services Interoperability Organization (WSIO). SOAP uses XML as the message format due its widespread use.

A SOAP message is an XML document that has an Envelope as the root of the document tree with two children: the optional header, a mandatory body. The header contains header informations, while the body contains the “procedure call” and the response of the procedure. The Fault element, when present, contains errors responses. Below is an example of a SOAP request for a remote procedure called “double” which supposedly doubles its argument. The root element is the Envelope which has two children: the Header and Body. We have used the optional header in this example to illustrate the use of intermediate node processing. Some intermediate nodes can choose to ignore some header fields especially if it does not implement them. In our example below a node cannot ignore the element importantField because it has the attribute must Understand set to 1. We will come back to this issue later. The Body contains the request for the remote procedure “double” that takes one parameter “x” in this instance given the value 7. The XML schema for the application is defined in some hypothetical domain “my.com”.

```
<?xml version=”1.0”?>
<soap:Enve lope
xmlns:soap=”http://www.w3.org/2003/05/soapenvelope”>
  <soap:Header>
     <myns:importantField xmlns:myns=”http://www.my.com/myns”
            soap:mustUnderstand=1>
           16
      </myns:importantField>
  </soap:Header>

  <soap:Body>
    <myns:double >
      <myns:x>7</myns:x>
    </myns:double>
  </soap:Body>
</soap:Envelope>
```

Listing 1. SOAP Request

In listing?? is the response from the server. The response illustrate the case when a header is not needed and is therefore omitted. The procedure “double” returns the value passed to it by the SOAP request.

```
<?xml version=”1.0”?>
<soap:Envelope
   xmlns:soap=”http://www.w3.org/2003/05/soapenvelope”>
 <soap:Body>
   <myns:doubleResult >
     <myns:x>14</myns:x>
   </myns:doubleResult>
 </ soap:Body>
</ soap:Envelope>
```

Listing 2. SOAP Response

The third listing below illustrate the case when an error occurs. For example if an invalid value, a string say, is passed to procedure double, an error is reported.

```
<?xml version=”1.0”?>
<soap:Envelope
 xmlns:soap=”http://www.w3.org/2003/05/soapenve lope”>
   <soap:Body>
     <soap:Fault>
       <faultcode>soap:Server</faultcode>
       <faultstring>invalid parameter</faultstring>
     </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

Listing 3. SOAP Fault

3. Web services’ security

The Web Services Security (WSS) standard by developed by OASIS [2] is a set of SOAP extensions that can be used to implement Secure Web Services that provide integrity and confidentiality. In addition, the specification given in [21] associates security tokens with message contents. It is designed to be extensible and supports multiple security token formats. The mechanisms provided by this specification allow to send security tokens (that are embedded within the message itself) in order to achieve message integrity and message confidentiality. The WSS standard, however, lacks two important security aspects: authorization and protection against Denial of Service attacks.

WSS specifies how to protect SOAP envelopes using a <Security> header containing security elements such as security tokens, which identify principals or sessions that could provide integrity and confidentiality:

1. In the case of integrity, parts of the envelope of the SOAP message could be protected by an XML digital signature.

2. In the case of confidentiality, parts of the envelope of the SOAP message could be encrypted, with additional XML references indicating how to retrieve the decryption key. The WSS standard defines several tokens. In this work we will consider only the username token[17].

4. Architecture

Our model is composed of three components: the challenger, the container and the policy module (See Figure 1). Each of these components in turn contains many parts. The web services container includes the environment for deploying web services as well as the integrity, confidentiality and authentication modules. Also, the policy module contains many submodules as will be discussed later.

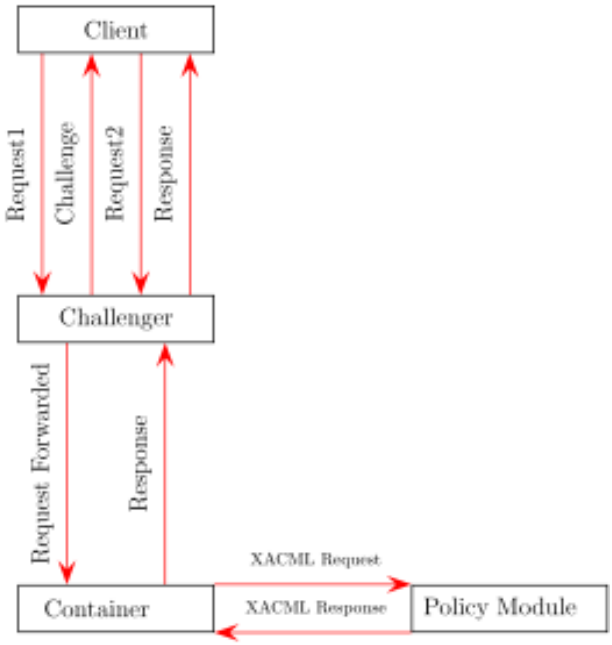

Figure 1. Model Architecture

1. The Challenger is the gateway that intercepts Web Service request and replies with a puzzle challenge on behalf of the server. It will also protect the system from Denial of Service attacks (DoS).

2. The policy module: this is where all the policies and access control is stored and processed.

3. The Web Services container: this is the server that hosts the web services and the basic integrity and confidentiality parts. It includes also the authentication service.

As we have seen, our model involves many components and technologies. We present these components and technologies next.

4.1 Policy Module

The eXtensible Access Control Markup Language (XACML) is a general purpose access control policy language defined using XML. It is designed to support the needs of most authorization systems. The main part of XACML defines the syntax for a policy language and the semantics for processing those policies. One of the main strong points of XACML is that one can abstract away the policy by defining a Policy Decision Point and using a request/response format to query the policy system. This allows us to develop the application logic independently of the policy system.

4.2 Container

The container serves is in the core of the architecture and provides multiples functionalities. First it is acts as a container for web services in a sense that all web service components are hosted by the container. On the other hand it provides the basic integrity and confidentiality capabilities using digital signatures and encryption. Also it provides user authentication using?? . All of these capabilities are implemented by software developed by the Apache organization.

4.2.1 Integrity and Confidentiality

The WSS standard is used among other things to "Protecting the message content from being disclosed (confidentiality) or modified without detection (integrity) are primary security concerns. This specification provides a means to protect a message by encrypting and/or digitally signing a body, a header, or any combination of them."[2] Even though there are many implementations of the WSS standard we choose the software developed by the Apache organization. Even the Apache organization has the standalone implementation, WSS4J [4] which is an open source java library that can be used to sign and verify SOAP messages. Many trusted web services building on top of the WS-security standard use the WSS4J framework.In our work we chose the Rampart module of the axis2 project. It supports both client and server applications.

Authentication is achieved through Rampart by using the "UsernameToken" that enforces the use of a username and a password.The username and password pair are protected inside the SOAP message by encryption.

4.2.2 Axis

The Axis2 container is the core engine for web services development build by the Apache organization [3]. In addition to the ability of adding web services interfaces to existing web applications, Axis2 can function as a stand-alone application server. Axis2 is used in this work to implement the role of the container in our model.

4.3 Challenger

The Challenger component protects the whole framework and specifically the container from Denial-of-Service attacks that pose a security threat especially for public web services. These attacks can come under different guises, bandwidth attacks and excessive work attacks. For the first type of attacks we use the method of tokens presented in [12]. The idea in this type of defense is to use the path signature of the request to prevent IP spoofing and once this is done we use a firewall to block attack packets. As for the second type of attacks we use the idea of client puzzles. While the original idea of cryptographic puzzles is due to [16], main papers have been published on the subject and used in different situations [11][9]. In this work we use the client version of puzzles where the server generates a puzzle to be solve by the client before the resources are allocated to the client. The main idea is for the generation of the puzzle to be much easier than solving the puzzle,i.e., the server should much less work than the client otherwise there is no benefit in deploying them. In this paper we use the version of client puzzles presented in??. In short, the server uses a hash function h the server sends the client the triple $(n, x', h(x))$ where x' is x with its n least significant bits set to zero. The client should return the solution to the puzzle, namely the value of x. Since the hash function is supposed to be one-way, the easiest way for the client to determine the value of x (the result) is try all 2^n possibilities, therefore doing much more computation than the server. In our implementation the hash function used is MD5.

5. Review of Related Work

There has been much work related to the security of web services. In this section we discuss some of that work. For a more comprehensive view see [8] and [20]. Dignum et. al. presented a model with a multilevel architecture for securing web services[10]. In [5] the authors show how to compute trust index of a service provider or a service requestor. They provide a method that continuously updated to reect change in behavior. Yamaguchi et al. [19] proposed an application programming model called "Web Service Security Application Programming Interfaces" to simplify the programming for end users who are not very familiar with WS-Security. Their model was based on the Service Oriented Architecture (SOA), the WS-security requirements, and on the existing APIs proposed by Microsoft. It consisted of six APIs that tend to achieve confidentiality and integrity through signatures and encryption.

Bhargavan et al. [6] addressed the problem of securing sequences of SOAP messages exchanged between web services providers and clients. Their work confirmed the inefficiency of using WS-Security alone for each message and that the integrity of a whole session as well as each message should be secured. They relied on WS-Secure conversation, which goal is to secure sessions between two parties, and on the WS-Trust, which describes how security contexts are obtained.

Gutierrez et al. [15] intended to describe a security model for web services in order to facilitate the development phase. Their model is based on web service security requirement specification, web service security architecture design and web service security standard selection. The research focused mainly on the web services security architecture.

Rahaman et al. [18] describe web services security architectures in a simplified way using WS standards, in addition to addressing the issue of attacking a SOAP message from XML rewriting attack. The research focused on message level security and discussed two different message flows that use (or do not use) SOAP message structure information.

Felix et al. [13] addressed the scalability and flexibility limitations of the WS authentication model where the acquirement of identity claims requires online interactions with security token services, thus introducing communication overhead and creating performance bottlenecks. They presented a new model where they addressed these limitations through two concepts: credentials for claim inference and claim-based issuer references. They showed how credentials are used both to increase the scalability and to reduce the number of online token requests. They also showed how the simultaneous usage of security tokens and credentials results in several advantages when compared to credentials used in trust management models.

Zhang [21] enumerated the main challenges that threaten a web service and make it "untrustworthy". He proposed a solution to address these challenges by adding an additional layer (i.e. WS-Trustworthy layer) on top of the WS- Security layer.

Garcia and Toledo [14] proposed a security model for web services that is based on semantic security policies. Their main goal was to ensure confidentiality and integrity of a SOAP message. The main components of the security model they designed are equivalent to some XML elements of the WS-Security, XML encryption and XML Signature standards. Note that they only addressed the issues of confidentiality and integrity.

To our knowledge, no single model addressed both authentication and authorization at the same time. We next give the description of our model that addresses these two security issues.

6. Implementation

In this section we describe the implementation of the model. The scenario that we will describe involves a client that needs to access a web service from a server.

1. The client sends a SOAP request to the server. This request is intercepted by the challenger. The response of the challenger depends on whether the request is public or private. If it is private then it contains an encrypted username/password combination so the challenger sends a puzzle challenge to the client. If the request is public then the challenger sends a token for the TCP connection as described in [12]. This prevents IP spoofing and "authenticate" the client.

2. The client has to solve the puzzle and resend the same request including the solution to the puzzle. This guaranteed that the client will do more work than the server and therefore minimizes the possibility of Denial of Service attacks.

3. The challenger forwards the request to the server.

4. The server will use the Rampart module to authenticate the request and once this is successful it will contact the policy module to ascertain that the request can be granted or denied. Once the server receives an answer from the policy module, and if it is positive than it will send the response to the challenger which in turn will forward the response to the client.

6.1 Performance

The gauge the effectiveness of our model we have performed a number of trials.Each trial consists of client request, one public and one private. The measurement took into account the authentication and authorization overhead as well as the overhead added by the challenger. As can be seen from the results when the number of users increase the average response is almost linear which is a good measure of the performance of the implementation.

Number Of users	Average Response
10	91
25	312
50	733
100	1489
500	7008

Table 1. Performance Evaluation

7. Conclusion

In this work we have proposed a solution to the problem of securing web services. Towards that end we presented a 3-tiered model that provides integrity, confidentiality, access control and protection from Denial of Service attacks. All the components that we have used in our implementation are open source components and conform to open and web standards. The confidentiality and integrity part are faithful implementations of the WS-Security standard, such as the security token "UsernameToken" in the authentication process. For access control we have used the extensible access control markup language (XACML) standard.The proposed model is flexible enough to host new security features in the future whether at the level of WS-security,

because we can add new to kens or at the application layer by integrating new entities to the data model. To our knowledge our model is the first that protects web services from denial of service attacks in addition to the other security features.

References

[1] SOAP version 1.2. "http://www.w3.org/TR/soap12-part1/".

[2] Web Services Security version 1.1. "http://www.oasis-open.org/committees/wss/".

[3] Apache axis2/java next generation web services. *(2008)In*: http://ws.apache.org/axis2/, 2008.

[4] The apache software foundation.(2008) *In:* http://ws.apache.org/wss4j/, 2008.

[5] Adam, N., Kozanoglu, A., Paliwal, A., Youssef, M. (2006). Mutual trust in open environment for cascaded web services. *In*: Proceedings of the 3rd ACM workshop on Secure web services, SWS '06, p. 107-108, New York, NY, USA. ACM.

[6] Bhargavan, Corin, Fournet, Gordon. (2007). Secure sessions for web services. *ACM Transactions on Information and System Security*.

[7] Box, Enhebuske, Kakivaya. (2000). Simple object access protocol. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[8] Damiani, E., Gabillon, A., Anderson, A., Staggs, D., Thuraisingham, B., Winslett, M. (2006) Directions and trends of xml and web service security. *In:* Proceedings of the 3rd ACM workshop on Secure web services, SWS '06, p. 1-2, New York, NY, USA. ACM.

[9] Dean, D., Stubbleeld, A. (2001). Using client puzzles to protect tls. *In:* Proceedings of the 10th conference on USENIX Security Symposium - V. 10, p.1-1, Berkeley, CA, USA. USENIX Association.

[10] Dignum, F., Dignum, V., Padget, J., Vazquez-Salceda, J. (2009). Organizing web services to develop dynamic, flexible, distributed systems. *In*: Proceedings of the 11th International Conference on Information Integration and Web based Applications & Services, iiWAS '09, p. 225-234, New York, NY, USA. ACM.

[11] Dwork, C., Naor, M. (1993) Pricing via processing or combatting junk mail. *In*: Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology, CRYPTO '92, p. 139 -147, London, UK. Springer-Verlag.

[12] Farhat, H. (2006). Protecting tcp services from denial of service attacks. *In*: Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense, LSAD '06, p. 155-160, New York, NY, USA. ACM.

[13] Felix, Pedro, Ribeiro. (2007). A scalable and flexible web services authentication model. *In*: Proceedings of the 2007 ACM workshop on Secure web services, p. 62-72.

[14] Garcia, D., De Toledo, F. (2008). Web service security management using semantic web techniques. *In*: CM symposium on applied computing, p. 2256-2260.

[15] Gutierrez, Fernandez-Medina, Piattini. (2008). Web services enterprise security architecture: a case study. Proceedings of the 2005 workshop on Secure web services, p. 10-19.

[16] Merkle, R., C. (1978). Secure communications over insecure channels. *Commun. ACM*, 21. 294-299.

[17] Nadalin, A. (2006). Web services security: Soap message security 1.1. http://docs.oasis-open.org/wss/v1.1.

[18] Rahaman, Schaad, Rits. (2006). Towards secure soap message exchange in a soa, *In*: Proceedings of the 3rd ACM workshop on Secure web services, p. 77-84.

[19] Yamaguchi, Chung, Teraguchi, Uramoto. (2007). Easy-to-use programming model for web services security, *In*: Proceedings of the The 2nd IEEE Asia-Pacic Service Computing Conference, p. 275-282.

[20] Yu, Q., Liu, X., Bouguettaya, A., Medjahed, B. (2008). Deploying and managing web services: issues, solutions, and directions. *The VLDB Journal*, 17. 537-572.

[21] Zhang, J. (2005). Trustworthy web services: actions for now. *In*: IT professional, p. 32-36.