A SRP Based Handler for Web Service Access Control

Flávio O. Silva, João A. A. Pacheco, Pedro F. Rosa, Ph.D.
Networking and Computing Architecture Group
Faculdade de Computação
Universidade Federal de Uberlândia
MG, Brasil.
{flavio, frosi}@facom.ufu.br , jaapacheco@ufu.br

Abstract

Security is a critical issue. The wide use of Internet applications has tremendous benefits, but a security fail can expose confidential data. Web Services is a software application that can be remotely accessed over Internet using interoperable standards based on XML. At this time, there are no broadly adopted specifications for Web Services security and in this paper we present an extension to the SRP (Secure Remote Password) protocol in order to apply access control to a Web Service as an independent layer. This additional independent layer can be built on any available Web Service platform. In this work we have used Apache AXIS for this end.

1. Introduction

Security is a critical issue. The wide use of Internet applications has tremendous benefits, but a security fail can expose confidential data.

Client authentication is a very used alternative to protect services and information, and a common way to perform this is using a password.

This approach, however, has its weakness and an attack can break this protection and the services could be accessed without authorization.

Web Services [1] provide a new use of the Internet where remote applications can communicate using interoperable standards. Using this approach, Internet can provide not only information, but also, another way of getting distributed computing.

However, Web Services security model is not yet fully defined and a lot of proposals are emerging, delaying the adoption of this technology in many situations.

Security is a layer in Web Services architecture and an environment for creating Web Services, such as Apache AXIS [2], does not have a way to authenticate its callers, so if the service is available, anyone can call it.

In this paper we present an extension to the SRP [3] (Secure Remote Password) protocol in order to apply access control to a Web Service as an independent layer. This additional independent layer can be built on any available Web Service platform. In this work we have used Apache AXIS for this end.

Section 2 introduces Web Services main concepts. Section 3 presents Apache Axis engine; Section 4 presents the SRP protocol. Section 5 describes the authentication control proposed and it's tests and results are shown in section 6. Section 7 presents some concluding remarks and potential future works.

2. Web Services Overview

A Web Service is a software application that can be remotely accessed using some protocols based on XML [4] (eXtended Markup Language).

Web Services standards define the format of the message, specify the interface to which a message is sent, describe conventions for mapping the contents of the message into and out of the programs implementing the service, and define mechanisms to publish and to discover Web Services interfaces.

SOAP [5] is a XML based protocol and it is the backbone to this new generation of cross-platform cross-language distributed computing applications named Web Services.

A SOAP message basically has the following format:

```
<soapenv:Envelope>
    <soapenv:Header>     </soapenv:Header>
    <soapenv:Body>       </soapenv:Body>
</soapenv:Envelope>
```

Figure 1. Basic format of a SOAP message

Proceedings of the 2004 IEEE International Conference on Services Computing (SCC'04)
0-7695-2225-4/04 $ 20.00 IEEE

The <soapenv:Envelope> envelope identifies the start and the end of the SOAP message. The <soapenv:Body> contains the message itself. Finally the optional <soapenv:Header> contains one or more XML elements, named header blocks.

Headers offer the mechanism by which SOAP can be extended to include additional features and functionalities to Web Service based applications, such as security, authentication, transactions, and other service level agreement attributes associated with the message.

Fault messages are generated if error occurs during SOAP messages exchange. In such case, the <soap:Body> element will contain only one element named <soap:Faut>.

Web Services protocols are based on XML standards; to get them secure XML is the natural language to perform this task.

A range of XML-based security mechanisms are needed to solve all security related requirements.

Authentication, which is one of the security requirements, consists of an entity prove to another the identity it asserts to have. Several methods can be used to perform this operation and the username/password is the most used. In the Web Services context entities are objects that communicate each other through SOAP messages.

Web Services based protocols, such as SOAP, are well defined and adopted. However, security standards are still being defined.

At this time, there are no broadly adopted specifications for Web Services security. As a result developers can either build up services that do not use these capabilities or can develop ad-hoc solutions that may lead to interoperability problems [8].

There are some initiatives to define standards for authentication in Web Service such as XML Key Management Specification (XKMS) [9]; Security Assertion Markup Language (SAML) [10] and WS-Security [10].

SAML specification, which is based in XML assertions, specifies the use of SRP for authentication, but until this time there is no implemented solution that uses SRP to authenticate SAML assertions.

Most of the WS-Security specification is based on SOAP header processing. This technique is the basic principle used throughout this work.

3. Apache Axis Engine

AXIS (Apache eXtensible Interaction System) essentially is a SOAP engine. It provides the infrastructure to construct applications and manipulate messages that uses SOAP.

AXIS is built on the concept of handlers. A handler is an object that performs a specific task. A handler chain is a special type of handler that can contain other handlers and chains.

AXIS framework consists of three chains - transport, global, and service which will process the message in that order, before reaching the service. Figure 2 presents AXIS Engine and their handlers and chains.

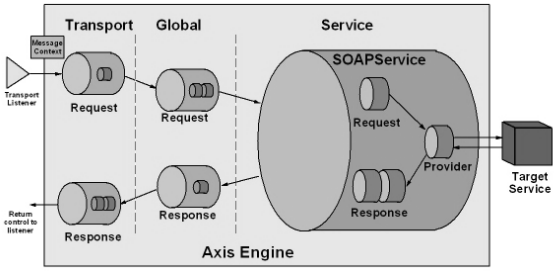

Figure 2. Axis handlers and chains

When the AXIS engine runs a set of Handlers are invoked in a predefined sequence. The particular sequence is determined by two factors: deployment configuration and whether the engine is a client or a server.

When a message is received, a Transport Listener will create a MessageContext [6] and will invoke the AXIS processing framework.

Theses handler chains can be configured and new handlers can be added modifying AXIS behavior.

4. SRP Protocol

SRP (Secure Remote Password) is one of the most used password-based authentication protocol. It provides a way to strongly authenticate a user without the usual risks of dictionary attacks faced by other password-based authentication schemes.

In this protocol the password is neither stored as a plain text nor in a ciphered way. Instead, a verifier, obtained from the password through a one-way hash function, is stored.

Another important characteristic of this authentication scheme is that the password is never sent across the network, thus avoiding that an intruder spoofs the network and retrieve the password or some information that could make possible a password reconstruction.

During the authentication process, ephemeral public keys are exchanged between the server and the client

and these keys are different for each authentication session.

Another important SRP assumption is that a user can choose a "weak" password without impacting the strength of the authentication scheme.

To perform the authentication a set of handshakes, between the server and the client must be accomplished. The diagram below shows each step of SRP protocol.

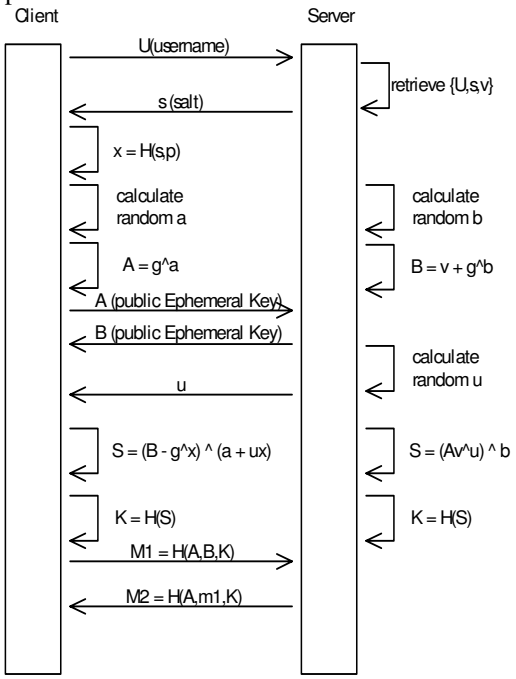

Figure 3. SRP protocol handshake

In first phase [3], the client sends to the server, its username (U). Then the server looks up in its database for the password verifier (v) and the client salt (s). The retrieved salt (s) will be sent across the network to the client. The client salt (s) is randomly generated and the verifier (v) is calculated when the password is stored in the database.

After receiving the client salt it computes a private key (x). This private key is calculated using a one-way hashing function (H) using as input the salt(s) and the password (p). Then the client generates the ephemeral private key (a) and the server calculates its ephemeral private key (b); both keys are generated randomly and not publicly revealed. Using the generator (g) and the ephemeral private key (a), the client calculates its public ephemeral key (A).

In order to exchange its private ephemeral key (B) with the client the server calculates it using the verifier (v) retrieved from the password database, the generator (g) and its private key (b).

Then the client and the server exchange theirs public ephemeral keys.

Using different methods and the public keys exchanged, both, the client and the server, calculates a session key (S), if the password is correct both keys will match.

Both sides hash the session key (S) into a cryptographically strong session key (K).

To show an evidence that the client has the correct session key, it computes the first match value (M1) and send it to the server. Then, the server computes this match value (M1) himself and verify if it matches with the client's value.

If values match, the server calculates the second match value (M2) as an evidence that he also has the correct strong session key.

Finally the second match value (M2) is received by the client. The client calculates the second match value himself and if both values are equal the client will be sucessfully authenticated.

Once the user is authenticated, an open secure session could be established for the communications.

5. Web Service Access Control

Before proposing the solution two basic goals were stated: first, the Secure Remote password protocol should be extended and converted to the Web Service way of work and second, the solution should be simple, and easy to be deployed, even with working Web Services that does not provides yet an authentication mechanism.

Considering the goals and the technical characteristics of SOAP, AXIS and SRP, the following solution was proposed:

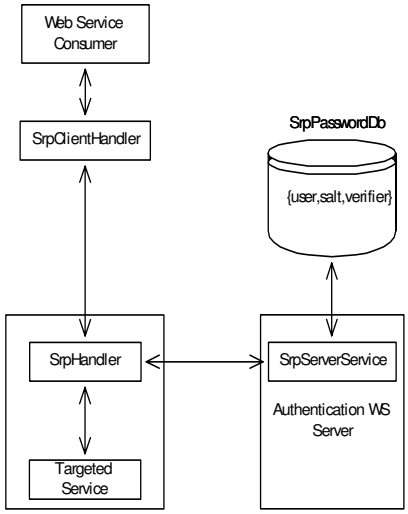

Figure 4. Authentication Handlers

In the solution proposed an AXIS handler (SrpClientHandler) will be added in the Web Service consumer message flow.

This handler will be part of the global request chain of this consumer, indicating that every time that this consumer requests a service, this handler will be executed before sending its request.

After being called, this handler will add in the original SOAP message a specific header block.

This header block will contain the username and the password that will be used in the authentication process, as shown bellow:

```
<soapenv:Header>
  <srpAuth:SrpAuthentication>
    <SprAuthenticationRequest>
      <user>flavio</user>
      <password>123</password>
    </SprAuthenticationRequest>
  </srpAuth:SrpAuthentication>
</soapenv:Header>
```

Figure 5. SOAP header used in authentication

An important assumption is that that this SOAP message will be transported on a secure channel. This can be easily achieved by using Secure Hypertext Transfer Protocol (HTTPS) in point-to-point communication between the SrpClientHandler and SrpHandler.

In the same way a handler will be added in the global handler chain of the targeted Web Service.

By doing this, every time this server receives a request, this handler, named SrpHandler, will be invoked before the SOAP request processing.

This handler will do the inverse process as it is the SrpClientHandler peer, in other words, it will catch the message and then it will retrieve the header block added by the peer, initially, in the message.

To perform this task SrpHandler will receive a MessageContext object. This object contains the modified SOAP message.

SrpHandler will verify whether the header block is valid and if it contains the necessary information to authenticate the client.

In the presence of error during this processing or invalid authentication information the handler will throw a <soap:Fault> message and the processing will be aborted.

Getting that the header is valid, SrpHandler will then starts the SRP protocol handshaking to verify whether the access will or not be granted.

In the proposed solution, the other peer, which will perform the server side of the SRP protocol, will be another Web Service, so this peer-to-peer communication will be done using SOAP messages, as well.

Figure 6 shows the messages that are exchanged between SrpHandler and SrpServerService.

The exchanged messages are the basis of SRP protocol and each time the handshake takes place different parameter values are used. This handshake cannot be reproduced elsewhere.

If the information initially received by the SrpHandler is correct the last handshake operation – verify(m2,m2_cli) – will succeed and the authentication handler will authenticate the user.

By doing this, the SOAP message that was intercepted by this handler will continue its message path and will reach the targeted service.

In the presence of problem during the authentication process SrpHandler will throw a <soap:Fault> and the original request will be blocked.

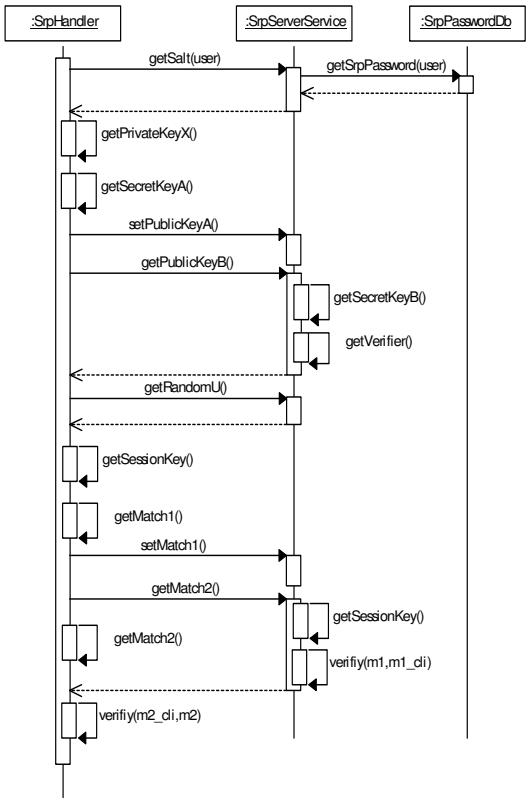

Figure 6. Authentication Web Service calls

The communication between SrpHandler object and SrpServerService shown in Figure 6 represents, indeed, Web Service calls. SrpHandler is located at the targeted Web Service and SrpServeService is located at another Web Service host.

To perform this operations the SRP protocol had to be extended, and its handshake is now performed

between these two Web Services using SOAP messages.

These services and the protocol primitives were defined using the WSDL (Web Service Description Language)[7].

These service definitions describe completely the Web Service and protocol primitives that are Web Service calls.

Although, the proposed idea was based on Apache AXIS architecture, the target Web Service can be any Web Service platform. I.e., this authentication scheme can be added to any Web Service, using an Apache AXIS Web Service as the authentication authority.

6. Handler Tests and Results

The proposed solution was deployed in order to verify if it was based on a correct approach and to test how this handler could be added to a running Web Service. It was deployed using the latest stable AXIS engine available.

To work with SRP protocol an object database had to be created. This database (SrpPasswordDb) contains objects that specify the authentication information used by SRP (see Figure 3). These objects contain the user name (U), the user salt (s) and a password verifier (v). This information will be added to the database when a new user is added to the system or when its password is modified. A simple application was created just for the database maintenance.

Table 1 shows how a "weak" password as "123" was stored in this database:

Table 1. Password stored information

Username (U)	flavio
Salt (s)	1495651668
Verifier (v)	79447583326133870766371702361095932362537850079379689849089590274344891658732515214436236014973517766023498861424977491439461982462467347278857120715151534537346952225746166317262858259530997403459108307088771145141425854549397348237012950496970919285196756046873246483705961449828584583135796507 92550023389

When a message arrives to the Web Services its handler (SrpHandler) starts the SRP protocol handshake. In the following lines the messages used to accomplish a successful authentication will be shown.

Initially the user salt is required, below is a simplified view of the request and response of this Web Service call:

```
<soapenv:Body>
  <ns1:getSalt>
    <user xsi:type="xsd:string">flavio</user>
  </ns1:getSalt>
</soapenv:Body>
```

Figure 7. GetSalt SOAP call request

```
<soapenv:Body>
  <ns1:getSaltResponse>
    <salt xsi:type="xsd:string">1495651668</salt>
  </ns1:getSaltResponse>
</soapenv:Body>
```

Figure 8. GetSalt SOAP call response

Then SrpHandler sends to the authentication Web Service (SrpServeService) the calculated public ephemeral key A:

```
<soapenv:Body>
  <ns1:setPublicKeyA>
    <A xsi:type="xsd:string">70527425795…</A>
  </ns1:setPublicKeyA>
</soapenv:Body>
```

Figure 9. setPublicKeyA SOAP request

Then this handler sends a request to retrieve the server calculated ephemeral public key B. The XML code below shows the server response SOAP message:

```
<soapenv:Body>
  <ns1:getPublicKeyBResponse>
    <B xsi:type="xsd:string">11518396...</B>
  </ns1:getPublicKeyBResponse>
</soapenv:Body>
```

Figure 10. getPublicKeyB response

The handler (SrpHandler) retrieves the random scrambling parameter (u). This value is used for both peers to compute the session key (S) and the strong session key (K), below is the server response:

```
<soapenv:Body>
  <ns1:getRandomUResponse>
    <u xsi:type="xsd:string">413525534</u>
  </ns1:getRandomUResponse>
</soapenv:Body>
```

Figure 11. getRandomU response

After the computing of the session keys, SrpHandler calculates the match value (M1) and sends this value to the authentication server:

```
<soapenv:Body>
  <ns1:setMatch1>
    <m1 xsi:type="xsd:string">132065658659...</m1>
  </ns1:setMatch1>
</soapenv:Body>
```

Figure 12. setMatch1 request

In this example the value (M1) of both the handler (Srphandler) and the server (SrpServeService) will match.

Then as a final step the handler request the second math value (M2) from the server.

The received second match value (M2) will then be compared with a second match value calculated by the SrpHandler and both values match, indicating that the authentication process occurred in a successful way.
At this time the handler (SrpHandler) will them finish its work and the original request sent by the client will continue its path way to the target service.

6.1. Latency time

The latency time of the proposed solution was compared to a system without authentication

Figure 13 shows the measured times. As expected SRP protocol introduces an extra processing time. This time was about 3.41 times higher.

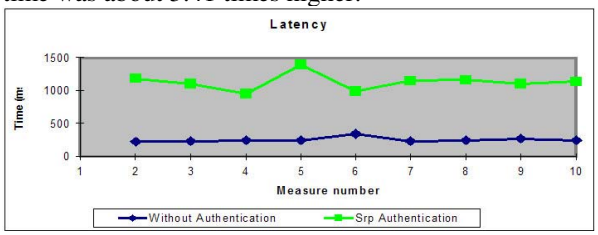

Figure 13. Latency time in milliseconds

In order to measure time required only for the authentication system proposed the network round-trip was not considered, so the same machine hosted the all the components showed in figure 4 – the client, the authentication server and the desired service as well.

7. Concluding Remarks

This paper presents a solution to get an effective access control to Web Services.

SOAP messages were used to extend SRP protocol, and, as far as we know, this is the first implementation of SRP over SOAP protocol.

The extension of SRP protocol presented is easy to be deployed and all the information was treated as a string element ("xsd:string"), thus, getting this solution with a high level of interoperability between the most distinct Web Services servers.

In this paper we have used SRP-3 [3] protocol to implement the handshakes. The latency time achieved in the round-trip of the service was almost 3.5 (three point five) times greater then the use of the system without the proposed solution. This result was expected concerning the extra processing added to the system. A right improvement to be done is use SRP-6 [11], which contains refinements in SRP protocol. In this version of SRP, the protocol handshake uses less web service calls, thus, the latency time tends to be shorter.

An important contribution is the fact that the authentication server is a Web Service as well, in this manner the solution can be easily added and reached in order to provides the access control to a Web Service and even to legacy applications.

The strong session Key (K) created by the SRP protocol can be used as well, to provide a session identifier in order to create a session control of an authenticated user.

Although the tests were made using Apache AXIS the client handler (SrpClientHandler) can be constructed to any available Web Service engine, leveraging security and making Web Service access control, an essential and critical feature, suitable to any existent deployed Web Service.

8. References

[1] Booth, D. et al. Web Services Architecture. World Wide Web Consortium Working Draft, 2003. [cited in April 22, 2004]. Available from: <http://www.w3.org/TR/2003/WD-ws-arch-20030808>.

[2] AXIS. Axis User's Guide. [cited in April 22, 2004]. Available from: <http://ws.apache.org/axis/java/user-guide.html>.

[3] Wu. T. The Secure Remote Password Protocol. In Proceedings of the Internet Society Network and Distributed System Symposium, pages 97-111, March 1998.

[4] Skonnard, Aaron. Essential XML Quick Reference: A Programmer's Reference to XML, XPath, XSLT, XML Schema, SOAP, and More, Indianapolis: Pearson Education Inc., 2002.

[5] Englander, R. Java and SOAP, California: O'Reilly & Associates Inc, 2002.

[6] AXIS. Axis Architecture Guide. [cited in February 22, 2004]. Available from: <http://ws.apache.org/axis/java/architecture-guide.html>.

[7] Christensen, E. et al. Web Services Description Language (WSDL) 1.1. World Wide Web Consortium Note. [cited in April 22, 2004]. Available from <http://www.w3.org/TR/wsdl>

[8] Booth, D. et al. Web Services Architecture. World Wide Web Consortium Note. [cited in April 22, 2004]. Available from <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211>

[9] Hirsch, F. et al XML Key Management (XKMS 2.0) Requirements. World Wide Web Consortium Note. [cited in April 22, 2004]. Available from <http://www.w3.org/TR/xkms2-req>

[10] Hartman, B. et al. Mastering Web Services Security, Indianapolis: Wiley Publishing Inc., 2003.

[11] Wu, T., "SRP-6: Improvements and Refinements to the Secure Remote Password Protocol", October 2002.