An Architecture for Secure SOAP Remote Procedure Calls

Adam Hess, Norman Jarvis, Ryan Jarvis, Kent E. Seamons, Quinn Snell
Computer Science Department
Brigham Young University
Provo, Utah, USA 84602
seamons@cs.byu.edu

ABSTRACT

The Simple Object Access Protocol (SOAP) is an XML-based messaging protocol used primarily as the wire format for a remote procedure call (RPC). SOAP offers advantages for RPC communications on the Internet. SOAP messages piggyback on HTTP and are able to penetrate Internet firewalls and reach distributed computing applications. SOAP also holds the promise of enabling interoperability between competing heterogeneous RPC-based systems. However, SOAP lacks standard security provisions for remote procedure calls. This paper presents Secure-SOAP, an open architecture leveraging existing web security technology that provides confidentiality, integrity, authentication, and authorization for SOAP RPC requests. The architecture includes a general-purpose authorization module based on the emerging Security Assertions Markup Language (SAML) standard. The authorization module can be invoked at a policy enforcement point just prior to the server invoking the remote method in order to enforce access control for SOAP RPC requests. The architecture demonstrates how any SAML-compliant authorization service acting as a policy decision point can be readily integrated with a SOAP RPC system to provide access control to SOAP RPC requests.

Keywords: SOAP; SOAP Security; SAML; XML; Remote Procedure Call

Technical Areas: Distributed systems security; Middleware and distributed platforms; Internet and web computing

1. Introduction

The rise of the Internet as a global network connecting millions of computers has led to a proliferation of distributed computing applications. Web users interact seamlessly with information integrated in real time from distributed sources. A remote procedure call (RPC) is an attractive programming model that application developers use to create distributed computing applications. RPC provides a convenient abstraction for remote services, making remote method invocations appear conceptually similar to well-known and well-understood local procedure calls. Some examples of existing RPC-based systems include DCOM, Java RMI, EJB, and CORBA.

There are two major roadblocks to deploying distributed applications across the Internet using existing RPC-based systems. First, the lack of interoperability between heterogeneous RPC-based systems hinders the rapid deployment of and access to web-based services. Second, traditional RPC-based systems have problems accessing distributed applications through Internet firewalls. Most firewalls limit the use of random ports because they are configured to accept connections to only a few ports, such as the standard HTTP port 80. This limits the ability to deploy distributed object protocols, like DCOM, that rely on dynamically assigned ports for remote method invocations.

The Simple Object Access Protocol (SOAP) is an XML-based messaging protocol [16] that has been submitted for standardization by the W3C and is being embraced by many vendors, including Microsoft and IBM. The primary advantages to using SOAP include ease of use, extensibility, platform independence, programming language independence, and ease of integration with Internet technologies. Although SOAP can be used with any transport protocol, it is most commonly embedded in HTTP—the standard protocol that powers the Web.

SOAP is primarily used as the wire format for a remote procedure call (RPC), the well-known technique that an application utilizes to execute parameterized methods or procedure calls on distributed machines over the Internet as

though on a single-machine. An RPC system includes the specification of the wire format for information that is transmitted on the network to and from a remote computer. An application invoking a remote procedure call encodes information into the wire format prior to transmission, a process known as *marshalling*. The receiver of the RPC request decodes the information from the wire format into a format supported by the receiver's local environment, a process known as *unmarshalling*. The process is reversed to return the result of an RPC to the application.

Suppose a web-based banking application uses SOAP to complete a remote procedure call to obtain an account balance. The application proceeds as follows. First, the client formats information regarding the procedure call and any arguments into a SOAP message and sends it to the server as part of an HTTP request. Figure 1 shows an example of an HTTP request with a SOAP message in the body of the request. Second, upon receipt of the RPC request, the web server translates the incoming SOAP message into a local method invocation to obtain the account balance for the given account. Third, after the local method call to obtain the account balance returns at the server, the server marshals the result of the RPC into a SOAP message. Fourth, the server transmits the SOAP response to the client in an HTTP response. Figure 2 shows an example of an HTTP response with a SOAP message in the body of the response. Finally, the client extracts the SOAP message from the body of the HTTP response and translates the reply from XML into the format needed in the client's application program.

```
POST /getAccountBalance HTTP/1.1
Host: www.acmebankservices.com
Content – Type: text/xml; charset="utf-8"
Content – Length: xxx
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
   xmlns: SOAP-ENV= "http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle=
   "http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
      <m:GetBalanceFromAccount xmlns:m="Some-URI">
         <AccountNum>111999000</AccountNum>
      </m: GetBalanceFromAccount xmlns:m="Some-URI">
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 1. A SOAP message for making an RPC request to get an account balance is shown embedded in an HTTP request.

```
HTTP/1.1 200 OK
Content – Type: text/xml; charset="utf-8"
Content – Length: xxx

<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle=
   "http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
      <m:GetBalanceResponse xmlns:m="Some-URI">
         <AccountNum>111999000</AccountNum>
         <Balance>100.00</Balance>
      </m:GetBalanceResponse xmlns:m="Some-URI">
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 2. A SOAP message containing the response from an RPC request to get an account balance is shown embedded in an HTTP response.

The use of SOAP for remote procedure calls provides a solution to the previously described roadblocks of deploying RPC applications on the web. SOAP is able to skirt the issue of Internet firewalls when embedded in HTTP because this allows SOAP messages to penetrate server firewalls that are configured to accept HTTP requests. SOAP also provides an open standard for a common wire protocol that can serve to enable interoperability between heterogeneous RPC systems, provided that they implement support for a SOAP-formatted RPC request and response.

Although SOAP offers advantages for RPC communications on the Internet, a principal drawback to SOAP is the lack of any security provisions for remote procedure calls, such as confidentiality, integrity, authentication, or authorization. The banking application example shown in figure 1 and figure 2 is not viable without these security facilities. This paper presents a comprehensive architecture for secure SOAP remote procedure calls that provides for confidentiality, integrity, authentication, and authorization. The architecture integrates SOAP with existing Internet security standards. This paper also describes an early prototype of the architecture and describes our plans for future research prototypes based on the design. This paper is organized as follows: section 2 presents the security architecture; section 3 discusses the implementation; section 4 includes related work; and section 5 contains conclusions and a discussion of future work.

2. SOAP RPC security architecture

The SOAP specification has no provisions for security. In this section, we present the design of Secure-SOAP, a secure RPC system providing integrity, confidentiality, authentication, and authorization. The design is sufficiently flexible to enable the integration of Secure-SOAP with a variety of existing web-based security technologies.

The Secure-SOAP architecture is shown in figure 3. The client and server communicate through remote procedure calls or remote method invocations using SOAP. When this communication occurs over an insecure network, the sensitivity of the RPC request may require confidentiality and integrity. Since a SOAP RPC piggybacks on existing communications protocols, such as HTTP, an architecture supporting SOAP RPC is able to leverage the security features of the underlying communications protocol. In order to provide confidentiality and integrity, the SSL protocol can be layered beneath HTTP to provide secure communication between the client and server.

The Secure-SOAP architecture relies on the server to authenticate clients making secure RPC requests. There are several alternatives for authentication, depending on the underlying communications protocol. Using SSL, the client can authenticate the server by requesting and verifying a server certificate. This guarantees the web client is communicating with the legitimate web server. SSL can provide for mutual authentication whenever the server requires that the client submit a certificate to the server. In practice, this is rarely done due to most clients lacking certificates entirely or lacking a certificate signed by an issuer that the server trusts for the purpose of client

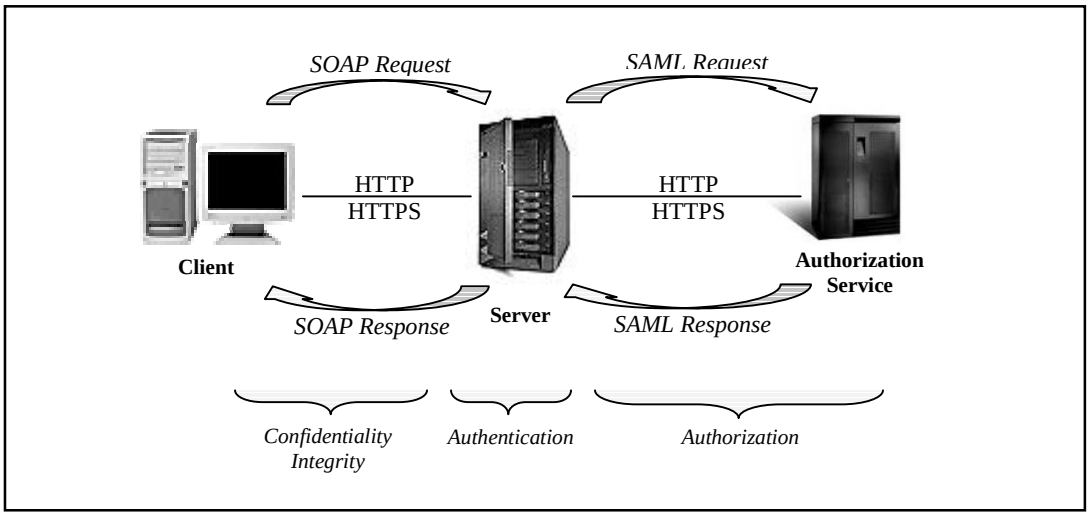

Figure 3. The Secure-SOAP security architecture providing confidentiality, integrity, authentication, and authorization for remote procedure calls using SOAP.

authentication. Thus, SSL can provide mutual authentication when both the client and server possess certificates. Alternatively, mutual authentication is typically accomplished by having the client authenticate the server in SSL through a server certificate, and then for the server to authenticate the client according to a username/password that are exchanged at the application layer following the establishment of an encrypted SSL connection.

When the HTTP protocol is used to transport SOAP, HTTP Basic or Digest authentication can be employed to authenticate the client based on a username and password. Basic authentication transmits the password across the network encoded in base64 format, which is trivially accessed by an attacker. Digest authentication does not transmit the password across the network. By combining Basic authentication with SSL, the vulnerability of Basic authentication is alleviated since SSL encrypts all application data during network transmission.

Authorization deals with the access control question of whether or not to grant permission to a certain subject to perform a given action on a particular object or resource. The Secure-SOAP architecture supports method-level access control granularity. To accomplish this, the server relies on an authorization service to determine whether or not access to the remote method or procedure should be granted. The server enforces the decision of the authorization service.

In the Secure-SOAP architecture, the server adopts the emerging Security Assertions Markup Language (SAML) [9] standard as the format for requests and replies to the authorization service. SAML defines an XML messaging format such that independent systems can exchange user identity and entitlement information in a standardized message format. SAML is being standardized at OASIS, the Organization for the Advancement of Structured Information Standards, and intends to incorporate portions of the S2ML [11] specification and the intended submission of AuthXML [1]. The SAML Protocols schema outlines a request/reply protocol. The requests make authentication, attribute, and authorization queries about a given subject. Secure-SOAP makes use of SAML authorization queries to the authorization service. A SAML authorization query asks one type of question: Should action(s) Y on resource Z be allowed for subject S given evidence E? In the context of SOAP, action Y is the method being invoked and Z is the class. The subject S is the principal making the RPC request and the evidence E is any other attributes or information about the user, such as the user's name, password, or other credentials.

The design of Secure-SOAP allows the server to communicate with any SAML-compliant authorization service for evaluation of an authorization query. The SAML specification refers to this as a *policy decision point*. Figure 4 shows an example of a SAML authorization query seeking permission for the subject `admin` to access a `deleteAccount` method.

Once the authorization service evaluates the SAML authorization query, a reply is returned encoded as a SAML response indicating whether or not the subject is granted access to the method. Figure 5 contains an example of the SAML response granting permission for the authorization request. The server is responsible for enforcing the decision of the authorization service. The SAML specification refers to this as a *policy enforcement point*. When access is granted, the server invokes the remote procedure in the usual manner. When access is denied, the server skips the method invocation and returns a response to the client indicating that access was denied.

```
<SAMLRequest RequestID="{some-id-value}"
   Version="http://www.oasis.org/tbs/1066-12-25/protocol/">
   <SAMLQuery>
      <Subject>
         <NameIdentifier>
            <SecurityDomain>acmebankservices.com</SecurityDomain>
            <Name>admin</Name>
         </NameIdentifier>
      </Subject>
      <Object>
         <Resource>BankAccount</Resource>
         <Action>deleteAccount</Action>
      </Object>
      <Evidence>
         <AssertionID>{some-id-value}</AssertionID>
      </Evidence>
</SAMLQuery>
</SAMLRequest>
```

Figure 4. A SAML authorization query requesting authorization to invoke the `deleteAccount` method.

```
<SAMLResponse ResponseID="{some-id-value}" InResponseTo="{some-id-value}"
   Version="0100" StatusCode="Permit">
   <Assertion type="saml:AuthorizationDecisionAssertionType"
      version=http://www.oasis.org/tbs/1066-12-25/
      Issuer="www.example.com" IssuerInstant="2001-05-31T13:20:00-05:00">
      <Conditions NotBefore="2001-05-31T13:20:00-05:00"
         NotOnOrAfter="2001-05-31T13:25:00-05:00"/>
      <Subject>
         <NameIdentifier>
            <SecurityDomain>acmebankservices.com</SecurityDomain>
            <Name>admin</Name>
         </NameIdentifier>
      </Subject>
      <Object>
         <Resource>BankAccount</Resource>
         <Action>deleteAccount</Action>
      </Object>
      <Evidence>
         <AssertionID>{some-id-value}</AssertionID>
      </Evidence>
      <Answer>Permit</Answer>
   </Assertion>
</SAMLResponse>
```

Figure 5. A SAML authorization response granting permission to invoke the `deleteAccount` method.

3. Implementation

An initial prototype implementation of the Secure-SOAP architecture presented in Section 2 is described in [8]. This section presents details of the implementation architecture, including design revisions from the lessons learned through developing and analyzing the system. Figure 6 illustrates the Secure-SOAP implementation architecture, demonstrating the steps to execute a secure SOAP RPC request.

The prototype client is a Java program supporting SSL server authentication and HTTP Basic authentication. The server is an integration of several software components, including the Apache HTTP server, Tomcat Servlet engine, and the Apache SOAP implementation. All of these components are freely available from the Apache Software Foundation at http://www.apache.org/. Client authentication in the prototype is handled using HTTP basic authentication over SSL and server authentication is provided using SSL server authentication.

The client and server communicate using SOAP over HTTP. Confidentiality and integrity of SOAP requests are provided using HTTP over SSL (HTTPS). When the HTTP server receives a SOAP RPC request, it is mapped to the SOAP context within Tomcat. Tomcat then receives the SOAP request and the request is handled by the rpcrouter servlet. At this point, a customized authorization check is inserted into the invocation path of the remote method invocations. The Apache SOAP implementation allows for custom handling of SOAP requests and responses. The ability to insert customized code in this manner is known as the Apache SOAP pluggable provider, which serves as an abstraction layer between the Apache SOAP server and the service implementation. Providers incorporate custom controls, such as access control checks, in the normal processing of the SOAP request and response. Each service includes a deployment descriptor that informs the Tomcat servlet engine of the designated provider to handle the SOAP RPC request for the service. The prototype includes a custom pluggable provider that performs an authorization check by calling a SAML authorization module prior to invoking the requested method. The authorization module sends a SAML authorization query to an authorization service that determines whether or not the RPC request should be authorized. The authorization service returns the result in a SAML response to the authorization module.

The prototype implementation utilizes YGuard [12] as the authorization service for making access control decisions. YGuard's modular design provides an extensible and efficient method of controlling access to system

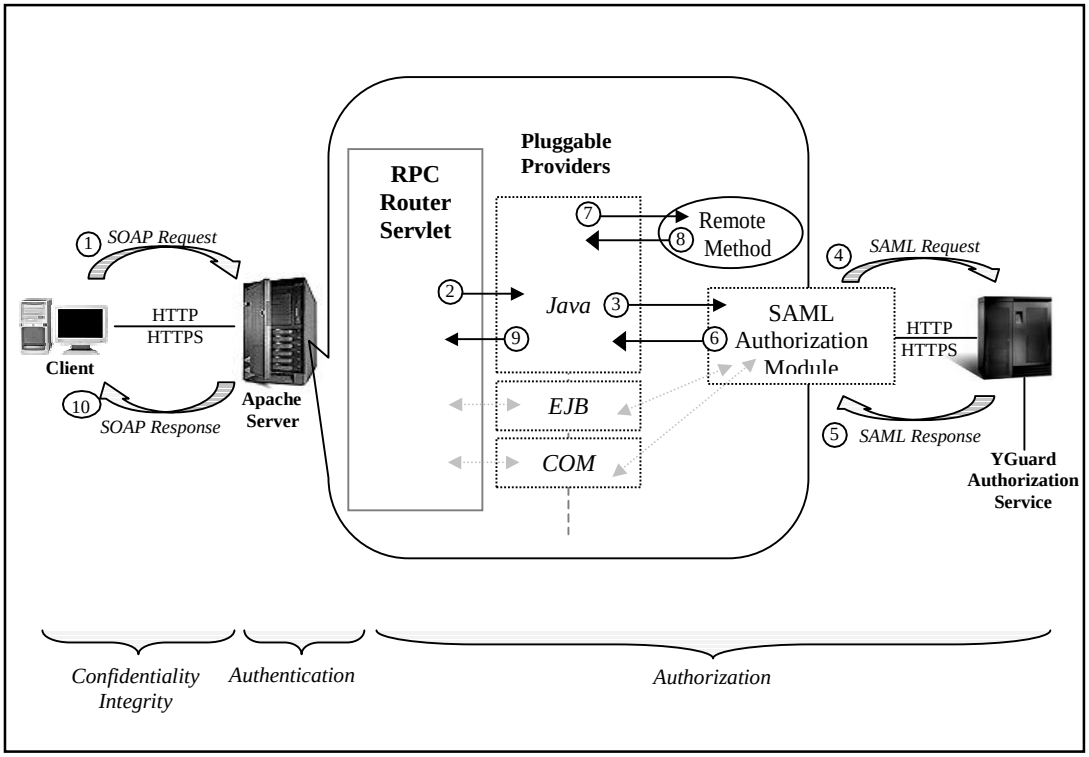

Figure 6. The Secure-SOAP implementation architecture, demonstrating the steps to execute a secure SOAP RPC request. The Apache pluggable provider calls the SAML authorization module that is responsible for formulating SAML authorization requests to authorization service. If permission is granted by the authorization service, the pluggable provider will invoke the remote method.

resources. YGuard supports access control policies specified using XML and employs set-based access control lists. This allows objects and subjects to be logically grouped so that a subject or group of subjects is authorized to access sets of objects. The core of this system is the set-access control lists (SACL) that allow permissions to be defined on sets of objects, extending traditional access control lists that only allow the permission to be defined for single objects. SACL implements a set representation modeled after XQL [10] that allows the addressing of object sets. The SACL structure forms a hierarchical permission tree where the nodes of the permission tree represent objects or set of objects to which permissions can be applied. Figure 7 illustrates a YGuard authorization policy for three methods of a hypothetical banking application.

In summary, the Secure-SOAP RPC architecture described in this section and shown in Figure 6 executes a SOAP RPC request as follows:

1. The client sends a SOAP request.
2. The rpcrouter servlet takes the SOAP request and passes control to the provider.
3. The provider calls the SAML authorization module prior to invoking the remote method.
4. The SAML authorization module sends a SAML authorization query to the authorization provider.
5. The authorization provider makes an authorization decision and replies with a SAML response.
6. The SAML Authorization module passes the decision to the provider.
7. If the authorization was successful, the method is invoked.
8. The invoked method returns.
9. The SOAP response is created.
10. The server returns the SOAP response to the client.

```
<YGuard mirror="root">
   <YGuard mirror="urn:BankAccount">
      <YGuard mirror="bankpackage.BankAccount">
         <YGuard mirror="deleteAccount">
            <SACL>
               <SACLEntry who="admin" thru="*"
                  operation="urn:Execute" when="*" xset="//*"/>
            </SACL>
         </YGuard>
         <YGuard mirror="getPreferredLoanRate">
            <SACL>
               <SACLEntry who="admin,user" thru="*"
                  operation="urn:Execute" when="*" xset="//*"/>
            </SACL>
         </YGuard>
         <YGuard mirror="getLoanRate">
            <SACL>
               <SACLEntry who="*" thru="*"
                  operation="urn:Execute" when="*" xset="*"/>
            </SACL>
         </YGuard>
      </YGuard>
   </YGuard>
</YGuard>
```

Figure 7. YGuard access control policy

4. Related work

There are a number of emerging XML security standards that could provide useful elements in a secure SOAP RPC architecture, including XML Signature (XML-DSIG) [13], XML Encryption (XML-ENC) [14], and the XML Key Management Specification (XKMS) [15]. The XML-DSIG standard specifies how to digitally sign XML documents and can be used to guarantee the integrity and authenticity of a SOAP RPC request. The XML-ENC proposal defines a means for the encryption/decryption of arbitrary binary data, an XML element, or its content and representing the result in XML. This can be used to provide confidential remote procedure calls. The Secure-SOAP architecture currently relies on SSL for confidentiality and integrity. XKMS defines XML based PKI operations consisting of registration, location, and validation. XKMS makes use of XML-DSIG and XMK-ENC and is additionally comprised of the XML Key Registration Service Specification (X-KRSS) and the XML Key Information Service Specification (X-KISS).

We are unaware of any available implementations of SAML. Netegrity has announced the release of JSAML, a Java SAML toolkit due to be released during the last half of 2001. Since SAML is relatively new, there is no practical experience available from systems supporting it. We intend to experiment with JSAML once it becomes available to determine whether it meets our needs.

GAA-API is a standard API for incorporating authentication and authorization into application programs [6]. Most of the authorization features of GAA-API will be utilized by the authorization service in the Secure-SOAP architecture. As authorization services using GAA-API become SAML-compliant, they will be able to plug into the Secure-SOAP architecture.

Some of the most common technologies incorporating RPC technology today include DCOM, EJB, and CORBA. Figure 8 provides a comparison of the communication protocol, network message format, programming language support, security model, and access control granularity that these technologies and Secure-SOAP provide. All of them employ a declarative security model where security can be provided without making changes to the application program. CORBA and SOAP have greater programming language independence compared to DCOM and EJB. Each technology employs a different communication protocol and network format. Secure-SOAP's

reliance on the standard web protocols provides a solution to the difficulty of penetrating network firewalls. Due to this and the open XML standard, SOAP promises to serve as a bridge between these other technologies to provide truly interoperable RPC technology for sending SOAP messages between EJB, CORBA and DCOM systems.

Criteria	DCOM	EJB	CORBA	Secure-SOAP
Security Model	Declarative	Declarative	Declarative	Declarative
Method Level Access Control	Windows custom security policies	Java Policies	priveleges, domains, required rights	YGuard
Communication Protocol	DCE-RPC	JRMP	IIOP	HTTP, FTP, SMTP, etc
Network Message Format	NDR	Java serialization	CDR	XML
Programming Language Support	Microsoft Specific	Java	Language Independent	Language Independent

Figure 8. A comparison of existing RPC-based technologies.

5. Conclusions and future work

This paper presents Secure-SOAP, an open architecture leveraging existing web security technology that provides confidentiality, integrity, authentication, and authorization for SOAP RPC requests. The architecture includes a general-purpose authorization module based on the emerging Security Assertions Markup Language (SAML) standard. The authorization module can be invoked at a policy enforcement point just prior to the server invoking the remote method in order to enforce access control for SOAP RPC requests. The architecture demonstrates how any SAML-compliant authorization service acting as a policy decision point can be readily integrated with a SOAP RPC system to provide access control to SOAP RPC requests.

A SOAP RPC has significant advantages when transmitted over HTTP on the web, since it overcomes the problem of gaining access to remote resources available through firewalls. SOAP RPC also overcomes the problem of homogeneous computing and programming environments by relying upon XML as its marshalling format.

The Secure-SOAP architecture provides security transparently, thus freeing the application developer from the complexities of implementing security directly. As the web continues to grow, web-based distributed applications will become the development paradigm of the future and the SOAP RPC security architecture will provide the security capabilities required by this new generation of web-based distributed applications.

In the future, we plan to extend the SOAP security architecture to incorporate ideas from our ongoing work in the area of trust negotiation, the exchange of digital credentials between strangers to gradually build enough trust to execute a secure transaction [2][7][18]. This will allow the Secure-SOAP architecture to provide authentication beyond simple usernames and passwords.

The current architecture focuses on using SAML only for authorization, but SAML also provides features for communicating authentication information. The architecture can be extended to incorporate SAML for client authentication between the client and server. As trust negotiation is integrated into the SOAP RPC architecture, we intend to explore SAML's capabilities to determine whether the current model meets the requirements for trust negotiation and propose extensions to handle any deficiencies. For example, trust negotiation allows for access control policies to be communicated to another party in order for them to dynamically discover the access control requirements for the secure resource they are attempting to access. In some cases, the client does not know a priori what credentials a secure service requires. Another area to explore is the scenario where a client requests SAML authentication assertions from a trusted third party in order to obtain the requisite credentials prior to invoking a secure SOAP RPC.

We will also explore ways to extend the system to support declarative security constraints on the values of method arguments. For some applications, such as the example in Figure 1 to obtain an account balance, the access control decision depends on the contents of the method arguments, not only on the method being invoked. For instance, permission to access the method to obtain a balance depends on an authenticated user requesting a balance for an account that they own. The YGuard system currently does not support access control decisions that depend on the content of method parameters. In a similar vein, our trust negotiation work includes client-initiated trust establishment in a server depending on the sensitive content of method parameters [2]. We intend to explore this issue further in the context of the Secure-SOAP architecture.

6. Acknowledgements

This research was supported by DARPA through AFRL contract number F33615-01-C-0336 and SPAWAR Systems Center San Diego grant number N66001-01-18908.

7. References

[1] AuthXML, "AuthXML.org", Official AuthXML web site, http://www.authxml.org/.

[2] T. Barlow, A. Hess, and K. E. Seamons, "Trust Negotiation in Electronic Markets," Proceedings of the *Eighth Research Symposium in Emerging Electronic Markets*, Maastricht, Netherlands, September 2001.

[3] A. D. Birrell, and B. J. Nelson, "Implementing Remote Procedure Calls," *ACM Transactions on Computer Systems*, 2(1): 39-59, February 1984.

[4] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP) 1.1," W3C Note 08, May 2000. Available at *http://www.w3.org/TR/SOAP*.

[5] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication," RFC 2617, June 1999.

[6] Global Operating Systems Technology Group, "Generic Authorization and Access control API (GAA API)," Information Sciences Institute of the University of Southern California. Available at http://www.isi.edu/gost/info/gaaapi/gaa_api.html

[7] A. Hess, J. Jacobson, H. Mills, R. Wamsley, K. E. Seamons, B. Smith, "Advanced Client/Server Authentication in TLS," to appear in the *Network and Distributed System Security Symposium*, San Diego, California February 2002.

[8] N. C. Jarvis, "SOAP RPC Security," Master of Science thesis, Department of Computer Science, Brigham Young University, June 2001.

[9] Organization for the Advancement of Structured Information Standards, "OASIS SSTC SAML Protocols Schema Discussion." Available at http://www.oasis-open.org/committees/security/docs/draft-sstc-protocol-discussion-01.pdf.

[10] J. Robie, J. Lapp and D. Schach. "XML Query Language (XQL)". Proposal to W3C QL98, September 1998. Available at http://www.w3.org/TandS/QL/QL98/pp/xql.html.

[11] Security Services Markup Language, "S2ML," Official S2ML web site, http://www.s2ml.org/.

[12] T. van de Akker, Q. O. Snell, and M. J. Clement, "The YGuard Access Control Model: Set-Based Access Control," Proceedings of the *6th ACM Symposium on Access Control Models and Technologies*, Chantilly, Virginia, May 2001.

[13] World Wide Web Consortium, "XML-Signature Working Group," W3C Notes on XML-DSIG. Available at http://www.w3.org/Signature/

[14] World Wide Web Consortium, "XML Encryption Working Group," W3C Notes on XML Encryption. Available at http://www.w3.org/Encryption/2001/

[15] World Wide Web Consortium, "XML Key Management Specification (XKMS)," W3C Notes on XKMS. Available at http://www.w3.org/TR/xkms/

[16] World Wide Web Consortium, "SOAP Version 1.2 Part 1: Messaging Framework," W3C Working Draft 2, October 2001. Available at http://www.w3.org/TR/2001/WD-soap12-part1-20011002/.

[17] World Wide Web Consortium, "Extensible Markup Language (XML)," W3C Notes on XML. Available at http://www.w3.org/XML/.

[18] T. Yu, M. Winslett, and K. E. Seamons, "Interoperable Strategies in Automated Trust Negotiation," Proceedings of the 8th *ACM Conference on Computer and Communications Security*, Philadelphia, Pennsylvania, November 2001.