



Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Security Essentials: Network, Endpoint, and Cloud (Security 401)"
at <http://www.giac.org/registration/gsec>

Web Services: Security Explained

GIAC Security Essentials Certification(GSEC)
Practical Assignment Version 1.4c

George Mao
March 23 2005

TABLE OF CONTENTS

I.	Abstract
II.	Introduction <ul style="list-style-type: none">1. What is a Web Service?2. How does a Web Service differ from other applications?
III.	Implementing a Web Service <ul style="list-style-type: none">1. Java2. .NET
IV.	The Security Risk
V.	Securing Web Services <ul style="list-style-type: none">1. Transport level encryption2. Message level encryption3. XML firewalls4. Authorization vs. Authentication5. The Building Blocks Of Security
VI.	Security vs. Performance
VII.	Outlook for Future Development
VIII.	Conclusion
IX.	Appendix
X.	List of References

© SANS Institute

I. ABSTRACT

The purpose of this guide is two-fold. First, it will discuss the recent development of Web Services Technology. This guide will provide background information on Web Services and then explain the inherent security risks that are associated with Web Services. Second, this guide will serve as a beginner's reference for Web Service developers to implement and secure their web services. This guide will enable readers to assess, for themselves, whether the advantages of Web Services outweigh the associated risk. Both Java and .NET Web Services will be discussed.

This document will show readers that Web Services can be secured, like any other web application. Web Service developers who follow well defined security techniques can successfully implement Web Services and use them to solve a wide variety of problems.

The intended audience is IT professionals, however a software development background will be beneficial in understanding the topics discussed here.

II. Introduction

Cnet.com reports "'Web services"--has the staying power to fundamentally change the way software companies do business and how people use the Internet (Wong, 1-3)."

If you've been following the IT industry, you'll know that the term "Web Services" has quite possibly been the biggest buzz word over the past couple of years. Web Services has the power to change the structure of internet communication. It has the potential to affect a wide range of people, from software developers to network engineers. It can also change the way organizations like insurance companies and banks share information. In the following sections this paper will show you what Web Services are, what they can be used for, and most importantly how to implement *secure* Web Services.

1. What is a Web Service?

A Web Service in its most generic term can be defined as a web application that has a *Service Oriented Architecture* (SOA). Sounds simple right? Yes and no. A Web Service is a collection of functions that are published to a network to be used by other programs or Web Services. These functions are strictly defined. They can be discovered and utilized through Internet protocols. In essence, a Web Service standardizes the means of communication among software applications. The World Wide Web Consortium (W3C) is the organization that is responsible for defining web service architectures. Their official definition is: "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages (Web service Architecture, 1.4)."

A common misconception about Web Services is that they are another form of web application. An easy way to differentiate web applications from services is to remember that applications usually have a graphical user interface (GUI) whereas a

Web Service does not. Users of web applications will directly access the program via some sort of interface, like a HTML web page. Users of a Web Service must “consume” a web service and implement their own interface to utilize the functionality. The term “consume” refers to the process a client takes to retrieve necessary information to use the service. A number of critical components need to be implemented before a client can “consume” a Web Service. The requirements are:

- A Web Service must support internet protocols, such as HTTP or FTP. All data will be transmitted via the Internet.
- Message data must be represented with the Extensible Markup Language (XML) specification. The XML message must follow the Simple Object Access Protocol (SOAP) specification (Walsh, 1).
- A Web Service should expose a description of its methods. This is used by clients to determine how to access these methods. This description should fit the Web Service Description Language (WSDL) standard (Introduction to WSDL, 1).

Later in this paper you will see how each of these technologies work and fit together in the scope of Web Services. There are many forms of implementation, however Java and .NET are the most popular and most supported development environments.

2. How does a Web Service differ from other applications?

Before Web Services were introduced, web applications tended to communicate with each other via many different propriety protocols. Some of these protocols may include Electronic Data Interchange (EDI), Distributed COM (DCOM), or Remote Method Invocation (RMI). For example, if you wanted to communicate with a web application you had to learn the specifics of its communication protocol. Next time you want to communicate with a different application you have to learn its protocol. In essence, each application could have its own protocol. This process becomes time consuming and repetitive. As a result, developers began to define ways to standardize data communication. A technology called Common Object Request Broker Architecture (CORBA) was eventually developed. This served as an intermediary for two applications that wanted to communicate. COBRA’s purpose is to receive traffic from communicating applications and translate the data into a common format. Thus interoperability was achieved. COBRA has begun to be phased out because it is complex and requires extra steps in the data transfer process. Web Services improves upon this by completely removing the intermediary (COBRA) layer. Communication is standardized between client and server. Next you will see how this is possible.

XML – Extensible Markup Language

Web Services are driven by XML. All communication is represented in XML format. Here is a sample XML message:

```
<message>
  <to>client</to>
  <from>server</from>
  <content>this is a message</content>
```

</message>

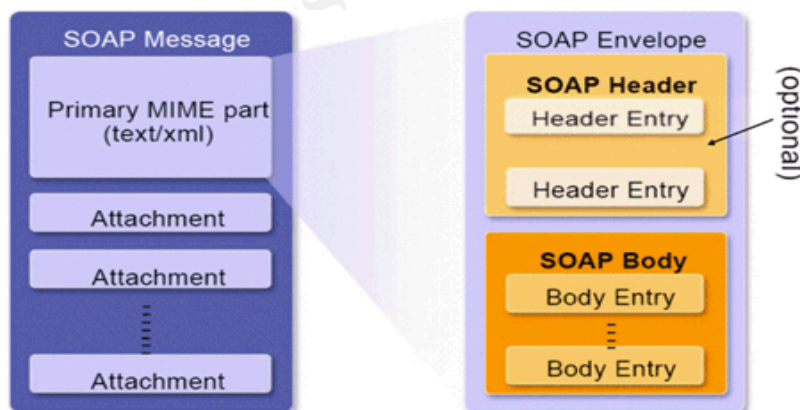
XML is used to describe data. The example above represents a message. It has “to” and “from” fields and the content of the message is defined within the <content> tag. XML is significant for two reasons. First, XML is not predefined. You are given the power to define the entire structure. This creates endless possibilities to represent data. Second, XML is represented in plain text. Plain text is important because it is the only universal format that can be understood across any operating system, software application, or web server. This allows XML to be sent from client to server or vice versa without any manipulation. A full XML tutorial is not in the scope of this paper, however you may wish to visit w3schools.com for more information.

SOAP – Simple Object Access Protocol

SOAP is the protocol Web Services use to exchange information over the internet. Many people confuse SOAP with internet protocols like TCP/IP or HTTP. SOAP does not deal with the transmission of the message, instead it is merely a “schema” for XML messages. A schema is a description of the structure, flow, and rules an XML message must follow. It can define everything from data types to content order. This is a crucial component because Web Services require messages to fit the SOAP specification. Without the correct format the message can not be understood. A basic SOAP message looks like this:

```
<SOAP-ENV:Envelope
  xmlns:
    SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  <SOAP-ENV:Header>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="some-URI">
      <Symbol>DEF</Symbol>
    </m: GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-Envelope>
```

The structure of a SOAP message is as follows:



(Mahmoud, 2)

As you can see, the SOAP message is made up of an Envelope. The Envelope holds

two objects, the Header and the Body. The Header is optional and the Body represents the actual content of the message. In the above example, a client is sending a SOAP message to a stock quote retrieval Web Service. The Web Service has a function called “GetLastTradePrice” which takes a “symbol” parameter.

As you can see, the SOAP body consists of a call to the “GetLastTradePrice” function. It also passes the DEF symbol as a parameter. Here is a closer look:

```
<m:GetLastTradePrice xmlns:m="some-URI">
  <Symbol>DEF</Symbol>
</m: GetLastTradePrice>
```

The underlying data format is XML. SOAP utilizes XML by defining its own tags and structure. Notice the tags in all caps: <SOAP-ENV:Envelope>, <SOAP-ENV:Header>, and <SOAP-ENV:Body>. All calls to Web Services must conform to the SOAP standard.

Sun Microsystems has written an overview of the SOAP specification. It is currently at version 1.2 and can be found here:

<http://java.sun.com/developer/technicalArticles/xml/webservices/>

WSDL – Web Service Description Language

The final component is WSDL (pronounced “wiz-dull”). WSDL is the specification for describing web services. A WSDL is a document that is usually auto generated specifying the exact method of communication the web service will follow. It defines data types, methods, and service location. All of this information is exposed through the WSDL. A client retrieves a WSDL and uses it to locate and invoke any available functions. The client does this by “binding” to the web service, which means generating specific classes that are defined and structured by the WSDL. Without a WSDL, there is no way for a client to know how to communicate with a service. Below is a shortened version of a WSDL, highlighting the main points. Note: The full example is located in the appendix (figure 1).

```
<?xml version="1.0" encoding="UTF-8" ?>
<wSDL:definitions targetNamespace="urn:sampleService" xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="urn:sampleService" xmlns:intf="urn:sampleService" ...>

  [...]
  <wSDL:portType name="sampleService">
    <wSDL:operation name="getSSN" parameterOrder="name">
      <wSDL:input message="impl:getSSNRequest" name="getSSNRequest" />
      <wSDL:output message="impl:getSSNResponse" name="getSSNResponse" /> </wSDL:operation>
    </wSDL:portType>
    <wSDL:binding name="sampleServiceSoapBinding" type="impl:sampleService">
      <wSDLsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
      <wSDL:operation name="getSSN">
        [...]
      </wSDL:operation>
    </wSDL:binding>
    <wSDL:service name="sampleServiceService">
      <wSDL:port binding="impl:sampleServiceSoapBinding" name="sampleService">
        <wSDLsoap:address location="http://localhost:8080/sampleService/services/sampleService" />
      </wSDL:port>
    </wSDL:service>
  </wSDL:definitions>
```

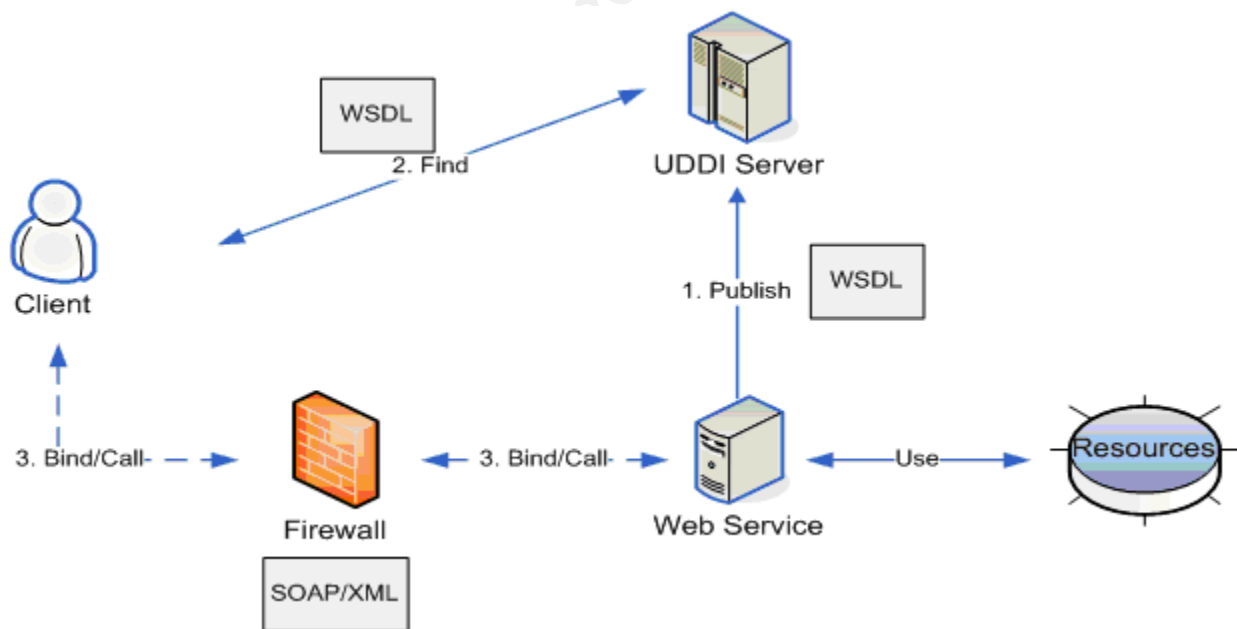
</wsdl:definitions>

Notice, the `<wsdl:operation name="getSSN">` tag. This tag defines the available operations for this web service. In this case, there is an operation called "getSSN." It takes an input parameter and also outputs the response. The other item worth noting in this WSDL is the `<wsdl:service name="sampleServiceService">` tag. This tag and the following items define the service, its name and the http location to the web service.

UDDI -- Universal Description Discovery and Integration

UDDI was designed because there was a need for clients to be able to dynamically discover and use a web service. In order to accomplish this goal, UDDI was developed to "house" a repository of information, describing any web service that registers with it. The purpose is to allow Web Services to publish the information necessary for clients to "consume" the service. A Web Service typically registers with a UDDI server by uploading its WSDL. To get a better idea of what UDDI servers accomplish, consider this: Phone books and white pages hold a wealth of contact information so that you, the client, can find this information and contact anyone who publishes to the phone book. A UDDI server serves the same purpose, except for web services. Clients looking for functionality will browse a UDDI server, find one that is useful, and retrieve information to use the service. The four main UDDI servers are hosted by IBM, HP, SAP, and Microsoft.

When XML, SOAP, WSDL, and UDDI technologies are combined correctly, a Web Service is created. The typical flow for using a Web Service is as follows:



1. Step 1. A Web Service locates and publishes itself to a UDDI Server by sending its WSDL. The UDDI server parses and stores the information. This step is very similar to adding an entry in the "white pages." At this point the UDDI server holds information describing the service for clients who want to "consume" the service.
2. Step 2. Clients who want to utilize a Web Service will browse a UDDI server, searching for services with desired features. When a client finds a usable Web Service the UDDI server will send the client the appropriate WSDL.

3. Step 3. Now that the client has the Web Service's WSDL, it knows how to communicate with the service. The client will use the WSDL to bind to the Web Service's functions. It also retrieves the Web Service endpoint from the `<wsdlsoap:address location>` tag. At this point, the client is ready to begin communications with the Web Service. All communication is exchanged via SOAP represented in XML.

Although UDDI technology can be extremely useful, security also is a primary concern. A malicious user can easily create a bogus Web Service and register it. This Web Service can mimic real functionality and pretend to be trustworthy. Unsuspecting users could potentially end up communicating with the bogus service without even knowing. All sorts of sensitive data could be at risk. This problem has been addressed by requiring business's to obtain authorization to register with the UDDI server before doing so. Approval will be granted by a UDDI operator after verification of certain credentials.

III. IMPLEMENTING A WEB SERVICE

Let's begin by showing an example of how to write and deploy a simple Web Service, both in Java and .NET. The prerequisite for running a Web Service is a web server and a SOAP engine. Below are the respective versions of a service that returns a social security number(SSN), given a name.

1. Java

In Java, we will use Apache Tomcat as our application server and Apache Axis as the SOAP engine. The following is the implementation for the SSN service.

Note: Tomcat and Axis must be installed and configured correctly for this to work.

```
sampleService.java:
public class sampleService{
    public String getSSN(String name){
        if(name.equals("Jaime"))
            return "111-11-111";
        else if(name.equals("Riyadh"))
            return "222-22-222";
        else
            return "Record Not Found!";
    }
}
```

2. .NET

In .NET, Microsoft Visual Studio.NET and IIS fill the prerequisites. This is a C# implementation for the SSN service.

```
sampleService.cs:
using System;
using System.ComponentModel;
using System.Web;
using System.Web.Services;

namespace sampleService{
    public class Service1 : System.Web.Services.WebService{
        [WebMethod]
```

```

        public string getSSN(String name){
            if(name.Equals("Jaime"))
                return "111-11-111";
            else if(name.Equals("Riyadh"))
                return "222-22-222";
            else
                return "Record Not Found!";
        }
    }
}

```

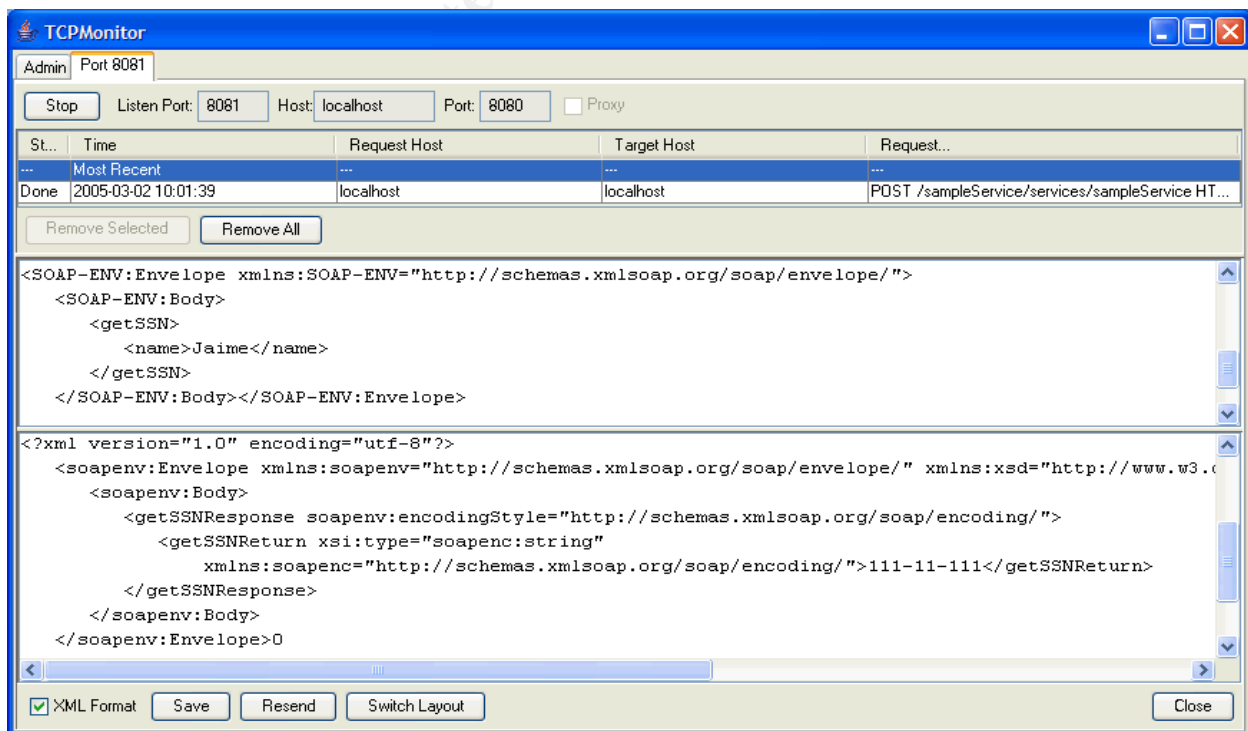
Both implementations define a “getSSN” function that returns a social security number(string) given a name(string).

IV. THE SECURITY RISK

Up until this point, this paper has primarily discussed the architecture of Web Services. This section will discuss the security risks. In the previous example, we designed and consumed a couple of simple web services that retrieved social security data. Now, we will inspect the actual message that is being transmitted across the wire, from client to server and vice versa.

The tool that is shown below is called TcpMonitor. It was developed by Apache as part of the Axis project. TcpMonitor is a utility that lets anyone watch the flow of SOAP messages between client and server. It intercepts and displays any information that was transmitted. We will use a person called Jaime as an example. He has a social security number of 111-11-111. First, we will send the above SSN service a request for Jaime’s social security number. Running the TcpMonitor tool on the example yields the following result:

Note: The data flow shown represents the Java version



Above is a screen shot of TcpMonitor after it has intercepted the SOAP call to getSSN(). The top pane is the SOAP message the client sent to the server and the bottom pane is the message the server returned to the client. By looking at the client side SOAP message in further detail, the SOAP structure can be seen again. There is an envelope, which contains a body, which in turn contains the call to the getSSN() function. The parameter that was passed in is the name, Jaime.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <getSSN>
      <name>Jaime</name>
    </getSSN>
  </SOAP-ENV:Body></SOAP-ENV:Envelope>
```

Now, by inspecting the server side SOAP message, we're able to see the data that was returned by the web service. Inside the SOAP body, there is a return parameter called *getSSNReturn*. This node holds the data a client will receive at the end of this web service call. In this case the value 111-11-111 was returned.

```
<?xml version="1.0" encoding="utf-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
      <getSSNResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <getSSNReturn xsi:type="soapenc:string"
          xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">111-11-111</getSSNReturn>
      </getSSNResponse>
    </soapenv:Body>
  </soapenv:Envelope>
```

Notice that all of the information is clearly visible because both client and server communicate using *clear text*. This is a default for web services communication. It gives the power to be widely interoperable and easy integration. However the disadvantage is that this creates a huge security risk. Imagine personal information or credit card numbers being sent over the wire like this. Hackers could have a field day if Web Services were left unsecured. Fortunately, there are many ways to secure a Web Service.

V. SECURING WEB SERVICES

As you saw in the previous section, without any form of protection, all of the communication between clients and web services can easily be exposed. Now you are probably asking yourselves, "how do we fix this security risk?" There are many ways to mitigate the risk. In the next section you will see how the following web service enhancements can make Web Services a secure platform for businesses.

1. Transport level encryption

Transport level encryption is the first layer of security for web applications. It is relatively easy to implement and normally results in acceptable performance loss. The most common transport level encryption is Secure Socket Layer (SSL) (Secure Sockets Layer, 1). SSL encrypts any message that is incoming or outgoing. SSL does not care about the contents of the message. The client and server establish an

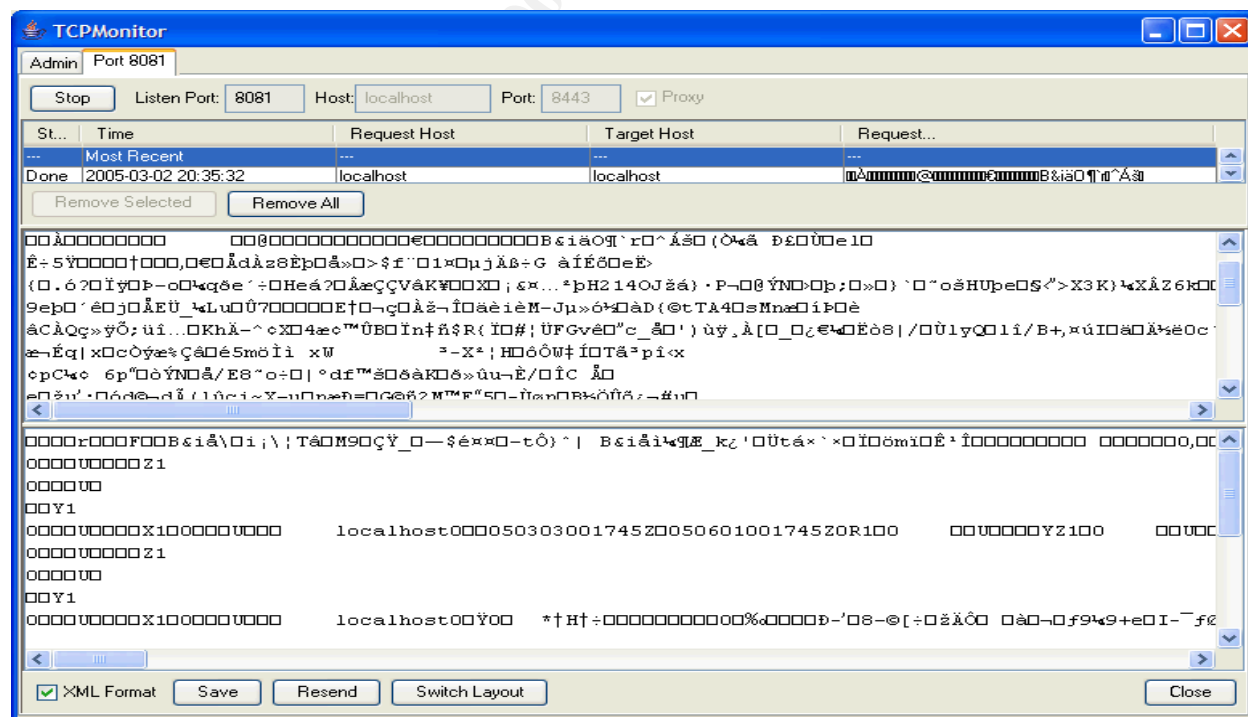
encrypted connection, based on public/private certificate exchanges. Once the message arrives at any endpoint, it is decrypted for use.

Enabling SSL is relatively easy, as it does not involve developing any code. There are two steps. First, obtain the certificates to use for encryption. Generally, this is done by purchasing one from a certificate authority such as VeriSign. However for purposes of this paper, we will self-generate the certificates. The second step is to enable the application server to use SSL and disable all other access. On the Java platform, this is a trivial task. Refer to Figure 2 in the appendix for examples of how to use the Java *Keytool* to generate a SSL certificates. Note: Java JDK 1.5 must be installed properly for this to work correctly.

The process in Figure 2 creates two certificates, server.keystore and client.keystore. They have each other's certificates embedded inside. The Web Service example shown in the previous section was executed on Apache Tomcat in clear text(no SSL). Enable SSL on Tomcat by creating a new listener, usually on port 443 or 8443 by adding this to the server.xml file:

```
<!-- Define a SSL HTTP/1.1 Connector on port 8443 -->
<Connector port="8443"
    maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
    enableLookups="false" disableUploadTimeout="true"
    acceptCount="100" scheme="https" secure="true"
    clientAuth="false" sslProtocol="TLS" />
```

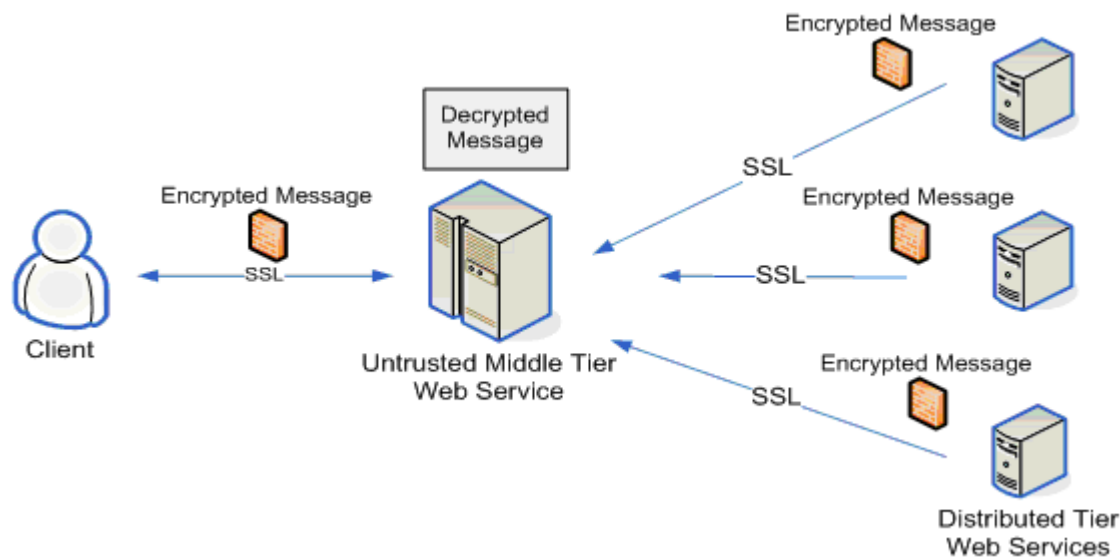
Next, place the server and client keystores in a location accessible by Tomcat. This enables SSL for the server. Now that SSL is enabled the results in TCPMonitor are very different. This is a screenshot of the data during transmission:



The data is completely encrypted, causing the SOAP messages to look like garbage. Sensitive data such as credit card information can be protected from hackers. This technique is relatively secure, assuming the certificates are not compromised.

However, there is a major weak link in this method. Consider this situation: You have a multi-tiered application. The client requests information from a centralized Web Service (middle tier). The middle tier forwards requests to various/distributed Web Services. The distributed tier returns data to the middle tier, who in turn forwards the data back to the client. If SSL was used as the security mechanism this is what would happen:

1. As the distributed tier returns data, it establishes a SSL connection and sends data to the Un-trusted Middle tier service.
2. The middle tier service will decrypt the SSL transmission, establish a new SSL connection with the requesting client, and forward the data.



There is a problem with this technique. In order for the middle tier to forward the data back to the client, it must gain access to the decrypted message. As a result, the entire message becomes visible. What if the data returned from the distributed tier is classified? What if the distributed tier does not want the middle tier to see the data? Or what if the middle tier is compromised by hackers? Transport level encryption works well, if none of these scenarios matter to the application. However, to guard against these scenarios the solution is to keep individual messages encrypted during the entire transmission process. In other words, the message should be kept encrypted along every hop. This is called Message level encryption.

2. Message level encryption

Message level encryption allows selective encryption of any or all parts of a SOAP message. This is beneficial for many reasons. First, encrypting the entire message can become very inefficient. If the message is large enough there will be a significant performance hit. Also, in most cases the entire message does not need to be encrypted, only small sections with the sensitive data require encryption. The alternative is to selectively encrypt sections of the SOAP message that contain sensitive data. Instead of encrypting the entire message, only the necessary portions of the message are encrypted. Performance is also increased because an SSL connection does not need to be negotiated between client and server. These differences can reduce the negative effect on performance if applied correctly.

Second, message level encryption allows the data to be kept encrypted along multiple

hops. There is no reason to allow an intermediary system to have access to sensitive data. In the Web Services context, Web Service Security (WS-S) is another name for message level encryption. The entire WS-S specification is currently being written by the Oasis organization (Web Services Security, 6). In order for encryption of this type to function correctly, many vendors have implemented their own APIs to support the WS-S standard. One such example is the Web Service Enhancements (WSE) API that Microsoft provides for the .NET environment (Web Services Enhancements, 1). This is an add-on for Visual Studio.NET that provides the capability to generate public/private keys, SOAP message encryption, and decryption.

Typically, a client request is sent to the server, along with the public key information embedded into the request. The server will use its key to encrypt selected portions of the message. Here is a snippet of a sample client request. As you will see below, there are extra tags that begin with “<wsse.” There are two tags, BinarySecurityToken and KeyInfo that contain the necessary information for encryption.

Note: The full request is located in the appendix as Figure 3.

```
<?xml version="1.0" encoding="utf-8" ?>
  - <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" >
[...]
```

```
  - <wsse:Security soap:mustUnderstand="1">
[...]
```

```
    <wsse:BinarySecurityToken ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
x509-token-profile-1.0#X509v3" EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
soap-message-security-1.0#Base64Binary" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-utility-1.0.xsd" wsu:Id="SecurityToken-98186c7d-1b3d-4bce-b260-
1c978b03e65a">MIIBxDCCAW6gAwIBAgIQxUSXFzWJYYtOZnmnuOMKkjANBgkqhkiG9w0BAQQFADAWMRQ
wEgYDVQQDEwtSb290IEFnZW5jeTAeFw0wMzA3MDgxODQ3NTlaFw0zOTEyMzEyMzU5NTlaMB8xHTAbBgN
VBAMTFFdTRTJRdWlja1N0YXJ0Q2xpZW50MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC+L6aB9x
928noY4+0QBxXnXkQE4quJl7c3PUPdVu7k9A02hRG481XlfWhrDY5i7OEB7KGW7qfJotLLeMec/UkKUwCgv3
VvJrs2nE9xO3SSWIdNzADukYh+Cxt+FUU6tUkDegg7dqwivOXhuOTryOI3HqbWTbumaLdc8jufz2LhaQIDAQA
Bo0swSTBHBgNVHQEEQDA+gBAS5AktBh0dTwCNYSHcFmRjoRgwFjEUMBGA1UEAxMLUm9vdCBBZ2VuY3
mCEAY3bACqAGSKEc+41KpcNfQwDQYJKoZIhvcNAQEEBQADQQAflbnMPVYkNNfXltG1F+qfLhHwJdfDUZu
PyRPucWF5qkh6sSdWVBY5sT/txBnVJGziyO8DPYdu2fPMER8ajJfI</wsse:BinarySecurityToken>
[...]
```

```
  - <KeyInfo>
  - <wsse:SecurityTokenReference>
    <wsse:Reference URI="#SecurityToken-98186c7d-1b3d-4bce-b260-1c978b03e65a"
ValueTypes="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" />
    </wsse:SecurityTokenReference>
  - </KeyInfo>
  - </wsse:Security>

  <soap:Body wsu:Id="Id-552ffa38-a006-4ba1-a49a-50cf382160ad">
    <StockQuoteRequest xmlns="http://stockservice.contoso.com/wse/samples/2003/06">
      <symbols>
        <Symbol>test</Symbol>
      </symbols>
    </StockQuoteRequest>
  </soap:Body>
</soap:Envelope>
```

The following is a snippet of the response from the server. Notice, the actual body of

the response message is replaced with the “<xenc:CipherValue>” tag. The encrypted data is wrapped with this new tag and the body of the message is no longer represented in clear text. Also notice that the rest of the SOAP message is not encrypted. For example, the SOAP envelope and header are clearly visible.

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" >
- <soap:Header>

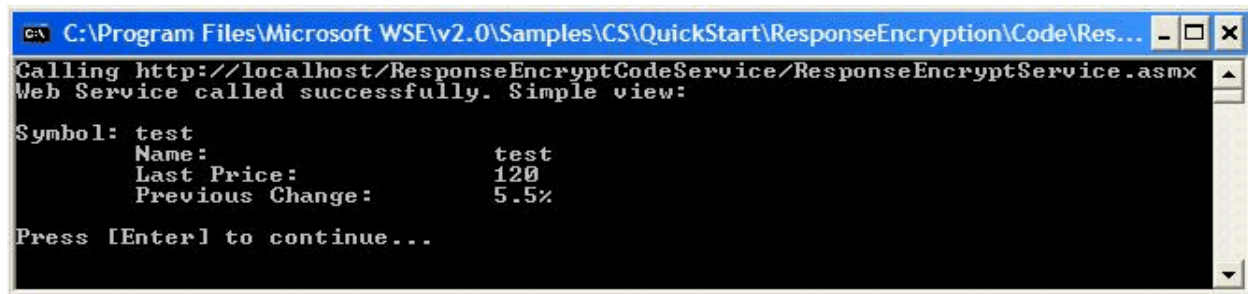
<wsa:Action>http://stockservice.contoso.com/wse/samples/2003/06/StockQuoteRequestResponse</wsa:Action>
  <wsa:MessageID>uuid:4e83fe61-ba1d-4b7a-88d0-9b7866e881bf</wsa:MessageID>
  <wsa:RelatesTo>uuid:8e9e1467-2596-476e-9b2a-933f1ae3c5be</wsa:RelatesTo>
  <wsa:To>http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous</wsa:To>
- <wsse:Security soap:mustUnderstand="1">
[...]
```

```
- <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
  <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
- <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
- <wsse:SecurityTokenReference>
  <wsse:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-
token-profile-1.0#X509SubjectKeyIdentifier">gBfo0147lM6cKnTbbMSuMVvmFY4=</wsse:KeyIdentifier>
  </wsse:SecurityTokenReference>
  </KeyInfo>
- <xenc:CipherData>

<xenc:CipherValue>hnNs2tWQE4J4nXEEjOBy/37lJeTTARr7tdnd+JhD/y67UTv537JKR+BGyWoQ3LezUmMEf
6fZOprdrhn+0vCSNSj4jLiiWV+L6AHujiBRv7Zsc6Agz2Fs9eyX+ugojXS3C0Xgr9z+PVGiW7+Yysm1/9ES2MF33
5qbGHZy2ZrAf3w=</xenc:CipherValue>
  </xenc:CipherData>
- <xenc:ReferenceList>
  <xenc:DataReference URI="#EncryptedContent-f4439e22-1ab9-446d-9a71-0a9a1d3363ce" />
  </xenc:ReferenceList>
  </xenc:EncryptedKey>
  </wsse:Security>
  </soap:Header>
- <soap:Body>
- <xenc:EncryptedData Id="EncryptedContent-f4439e22-1ab9-446d-9a71-0a9a1d3363ce"
Type="http://www.w3.org/2001/04/xmlenc#Content" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
  <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc" />
- <xenc:CipherData>

<xenc:CipherValue>hbn2rTnaZx8AhQBSkZWxH5lB+Dcnu4bm8mcUfHAZtfZCbKjF6TbsiPk7FcaOG92EoGEF
Fitxd6BSLZD5G+McpQ7409jA8xr3qXHkM6Tdwr6BuBG/IEGC2dH3OyOgYhcvtzdZj1V3nEvIv4fCkkA5wfjIN/eR
PLfks5wjFOZtMnRilEk2J2Q5ny4UpH9O0+NI0dBzhzhei3CYVuyZtEilcDbWu2byh6ZzH3/u02hgLasFy8hQDYNs
Sd61YmM5tgyfDCauL3VW8DrGNuHGwPFBtg81z6LgxDjiICTiPjGNNtXZF6LFEYNzdsIMKMoaf4ZWuXMGk
6d+z8VcosivXtwOrT83HBoAkstVdaSYivq62ET3DHRJYUqX8iiQvUoLqs0GQPDaewPuZrEQdiVrhmb2hvw8s7c6
lmhAC62DF4uGSpMYJgCR9ri2Tcbf/3WdWBVSU0U1a2mQyTcf5QOCfpcjZ0zXjdPyImITi1iAeeeZOkbD33+ncK/
X/7rvFXRZ59/nNtZjrTCL4pR57ryGeOLP7nATkD53cheKPtekgOkECGYu1sODw5BuFpYmIgpGs/D6lFrkNIHVp
FEkpRdu6MQ==</xenc:CipherValue>
  </xenc:CipherData>
  </xenc:EncryptedData>
  </soap:Body>
</soap:Envelope>
```

After receiving the server response, the client decrypts the message with its public key and has access to the data:



```
C:\Program Files\Microsoft WSE\v2.0\Samples\CS\QuickStart\ResponseEncryption\Code\Res...
Calling http://localhost/ResponseEncryptCodeService/ResponseEncryptService.asmx
Web Service called successfully. Simple view:
Symbol: test
      Name:      test
      Last Price: 120
      Previous Change: 5.5%
Press [Enter] to continue...
```

It is possible to use Transport and Message Level encryption together. An example of this is establishing SSL connections and using WS-S encryption simultaneously.

3. XML Firewalls

Encryption is a fundamental component of a good security plan. However, the other fundamental component is the prevention of adverse inbound and outbound traffic. This is normally done with hardware or software firewalls. Firewalls can block traffic on many exposed ports, thus preventing damage from vulnerabilities. These traditional firewalls are called “packet-level” firewalls because they operate on a very low level by inspecting packets, ports, and IP addresses.

Web Services create an entirely new problem that can not be overlooked. Web Service communication typically happens over HTTP, which is port 80 (by default). The data is sent via XML/SOAP. XML and SOAP easily pass through traditional firewalls because HTTP traffic on port 80 usually cannot be blocked. (Blocking port 80 will limit internet functionality so this usually not an option.) Finally, XML/SOAP and the HTTP protocol operate on a much higher level than “packet-level” firewalls can inspect. The solution to this security issue is a new type of firewall called an XML firewall.

XML Firewalls work by validating individual messages sent and received by Web Services. All XML content including messages, data elements, and function calls are inspected. XML firewalls determine if the client has access to anything it is requesting. By using XML firewalls, you can detect and prevent unauthorized access that can not be detected by traditional firewalls. XML-based attacks can be detected with this method (Smith, 1-3). Note, XML firewalls can not function if they are not configured correctly. For example, if the incoming traffic is SSL encrypted, the XML firewall must be configured with the correct certificates to decrypt and analyze the traffic. When message level encryption is used, there may not be a way to filter the encrypted portion; however, SOAP validation can still be done.

One of the leading products is *FireWall-1*, created by Check Point Software. The principle behind firewalls is to block unauthorized users or attacks to your service before they have a chance to cause any damage. If the attack never reaches your service, then it cannot cause harm. The drawback of the XML firewall solution is that it is usually expensive and very complex in terms of setup and configuration.

A second option is to create a custom “application-level” firewall. These firewalls sit between a “packet-level” firewall and the actual application. They are usually

implemented with software and work by filtering all incoming messages. All incoming traffic must first pass through the “packet-level” firewall. Then the message is directed to the “application-level” firewall before being processed by the real application. These firewalls check for valid user information, certificates, or other credentials. They also filter out potential harmful traffic. Once validated, it forwards the request to the requested Web Service. Security Assertion Markup Language (SAML) is often used in this process. The OASIS organization is leading the effort to create the SAML standard (Technical Overview of SAML, 3-7).

4. Authentication vs. Authorization

Authentication vs. Authorization is a topic that should be kept in mind. It is easy to confuse these two aspects of security. Authentication is the process that takes place to verify a user’s credentials. Requesting login information and valid certificates are such examples. Authorization takes place to determine if a user has been granted access to the requested service or parts of the service. Typically, authorization takes place after a user has been authenticated because the service trusts the user and now must determine if the user has access to specific resources.

As mentioned previously, SAML is often used in this process. SAML is a specification based on XML. Its purpose is to define a schema for exchanging authentication and authorization data. Much like SOAP, SAML defines a standard for formatting XML tags. However, SAML is focused on providing security related information. SAML’s basic principle is to provide “assertions” about a “subject.” An assertion is a declaration of facts in relation to the subject. There are three main types of assertions: authentication, attribute, and authorization. These assertions declare information such as user name, company, or resource requests. The basic flow for a SAML assertion is as follows: A subject asks for access to a resource. The subject provides some sort of evidence. The evidence could be a password, digital signature, or simply a user name. The server that understands SAML receives this request, uses this evidence, and verifies it to determine if the requestor has access to the requested data. The following is a sample authentication assertion. It is asserting that the subject is joeuser, who is represented in the Company.com domain. He has been authenticated by Company.com and the authentication method used was a password. The authentication time was March 21st, 2002.

```
<samlp:Assertion
  MajorVersion="1" MinorVersion="0"
  AssertionID="128.9.167.32.12345678"
  Issuer="Company.com"
  IssueInstant="2002-03-21T10:02:00Z">
  <saml:Conditions
    NotBefore="2002-03-21T10:02:00Z"
    NotAfter="2002-03-21T10:07:00Z" />
  <saml:AuthenticationStatement
    AuthenticationMethod="password"
    AuthenticationInstant="2002-03-21T0:02:00Z" >
    <saml:Subject>
      <saml:NameIdentifier
        SecurityDomain="Company.com"
        Name="joeuser" />
    </saml:Subject>
  </saml:AuthenticationStatement>
```

</saml:Assertion>

(Introduction to SAML, 18)

The attribute assertion is slightly different. Its purpose is to assert that a subject is associated with certain attributes. For example, the subject joeuser can be associated with the attribute “Age” that has a value “22.”

```
<saml:Assertion ...>
<saml:Conditions .../>
<saml:AttributeStatement>
  <saml:Subject>
    <saml:NameIdentifier
      SecurityDomain="Company.com"
      Name="joeuser" />
  </saml:Subject>
  <saml:Attribute>
    <saml:AttributeDesignator
      Attributename="Age" />
    <saml:AttributeValue>22</saml:AttributeValue>
  </saml:Attribute>
</saml:AttributeStatement>
</saml:Assertion>
```

(Introduction to SAML, 20)

The final assertion type is an authorization assertion. These assertions grant or deny access to the requested resource using some type of provided evidence. This example has a subject, joeuser. He has been granted “execute” permissions on the “login.jsp” resource.

```
<saml:Assertion ...>
<saml:Conditions .../>
<saml:AuthorizationDecisionStatement>
  Decision="Permit"
  Resource="login.jsp">
  <saml:Actions>
    <saml:Action>Execute</saml:Action>
  <saml:Subject>
    <saml:NameIdentifier
      SecurityDomain="Company.com"
      Name="joeuser" />
  </saml:Subject>
</saml:AuthorizationDecisionStatement>
</saml:Assertion>
```

(Introduction to SAML, 22)

5. The Building Blocks Of Security

All of the previously discussed techniques pertain directly to Web Services. These are not the only security techniques that can be used to mitigate security risks. Standard techniques still apply. It is important to understand the four main principles of security.

- Confidentiality – This refers to the idea that data transmitted from one point to another must be hidden or unuseable by third parties. Encryption is a trusted solution for this requirement. SSL and WS-S are examples.
- Integrity – This means that any changes to the original data can be detected. It allows two communicating parties to be absolutely certain that no one is

tampering with their data. Digital signatures and hashing algorithms such as SHA-1 are examples.

- Nonrepudiation – This principle means the sender of the message cannot deny that they sent the message.
- Authentication/Authorization – As discussed in section V.4.
- Availability – This final principle not only refers to the ability to provide critical information upon request but it also means denying availability to harmful attacks. All of the firewalls discussed earlier aid in this process.

Also remember to design services that guard against typical attacks like SQL Injection, Cross-Site scripting, and Denial of Service. Input validation is also important. Good coding practice should always be used. Do not reveal stack traces and unnecessary log messages to the user. Finally, never grant full administrative access to any user that does not explicitly require it.

VI. SECURITY VS. PERFORMANCE

Security is an essential component in any software product, however it must be applied in reasonable amounts. The major drawback to security is the performance degradation it causes. Any time security is integrated into an application it will cause a slow-down in transaction time. In the business world, time is money, so balancing security with performance is a very important task. *Software Test & Performance Magazine* reports that the use of SSL is usually 60-70% slower than normal HTTP (Joung, 22-29). A request that takes 5 seconds to process in normal HTTP, would be much slower with SSL enabled. If that request were 60% slower it would take 8 seconds. You might be thinking, how much impact does a couple of seconds have on anything? Pretend you are an online stock trader. You initiate an order to buy 10,000 shares of XXX company at \$50 each. You plan to spend \$500,000. The online broker guarantees you a 5 second purchase time. They enable SSL and performance takes a hit, causing the order to take 8 seconds. During those 3 extra seconds the stock price goes up by 50 cents. You just purchased 10,000 shares at \$50.50 instead of \$50.00 and spent \$5000 more than you planned to.

This may be an extreme example, but nevertheless you must fully test and understand your security enhancements. Always proceed with caution when implementing security mechanisms. Never assume a web service with enhanced security will result in the same performance efficiency.

Finally, just because security complicates performance issues does not mean security problems are dead ends. For example, there is a solution to SSL's performance degradation. There are many vendors who sell hardware accelerators for SSL, such as ArrayNetworks and NetScaler™. If applicable, these can be put to use.

VII. OUTLOOK FOR FUTURE DEVELOPMENT

Web Service technology has already matured to point where it can be used by major businesses. However, the technology is constantly evolving and changing. For instance, OASIS has recently completed SAML version 2.0 specifications. A new

version of WS-S is also on the horizon and the UDDI 2.0 specification is very near completion.

VIII. Conclusion

This guide has discussed the evolution of Web Services and why they are in existence today. It has explained why Web Services were created and the kind of problems they are meant to solve. Basic Web Service architecture and fundamental components were discussed, including XML, SOAP, WSDL, and UDDI. Finally, the pros and cons of security techniques such as SSL, WS-S, XML Encryption, and SAML were evaluated. Hopefully, this paper has given you a starting point to *securely* implement your own Web Services. Obviously, Web Services are not perfect, but they are evolving and could become the solution of choice, for communication over the internet. I hope you will be able to use some of the information provided here to start your own Web Service endeavors!

IX. APPENDIX

Figure 1 – Sample WSDL for getSSN Web Service

```
<?xml version="1.0" encoding="UTF-8" ?>
<wSDL:definitions targetNamespace="urn:sampleService" xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="urn:sampleService" xmlns:intf="urn:sampleService"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
  xmlns:wSDLsoap="http://schemas.xmlsoap.org/wSDL/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!--
    WSDL created by Apache Axis version: 1.2beta
    Built on Apr 19, 2004 (01:53:00 EDT)
  -->
  <wSDL:message name="getSSNRequest">
    <wSDL:part name="name" type="soapenc:string" />
  </wSDL:message>
  <wSDL:message name="getSSNResponse">
    <wSDL:part name="getSSNReturn" type="soapenc:string" />
  </wSDL:message>
  <wSDL:portType name="sampleService">
    <wSDL:operation name="getSSN" parameterOrder="name">
      <wSDL:input message="impl:getSSNRequest" name="getSSNRequest" />
      <wSDL:output message="impl:getSSNResponse" name="getSSNResponse" />
    </wSDL:operation>
  </wSDL:portType>
  <wSDL:binding name="sampleServiceSoapBinding" type="impl:sampleService">
    <wSDLsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
  </wSDL:binding>
  <wSDL:operation name="getSSN">
    <wSDLsoap:operation soapAction="" />
  </wSDL:operation>
  <wSDL:input name="getSSNRequest">
    <wSDLsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:sampleService"
      use="encoded" />
  </wSDL:input>
  <wSDL:output name="getSSNResponse">
    <wSDLsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:sampleService"
      use="encoded" />
  </wSDL:output>
</wSDL:definitions>
```

```

        use="encoded" />
    </wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="sampleServiceService">
<wsdl:port binding="impl:sampleServiceSoapBinding" name="sampleService">
<wsdlsoap:address location="http://localhost:8080/sampleService/services/sampleService" />
</wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

Figure 2 - SSL Certificate Generation using Java Keytool

Generate the server keystore and save in server.keystore:

```
keytool -genkey -alias tomcat-sv -dname "CN=localhost, OU=X, O=Y, L=Z, S=XY, C=YZ" -keyalg RSA -keypass changeit -storepass changeit -keystore server.keystore
```

Export the certificate from server.keystore to server.cer

```
keytool -export -alias tomcat-sv -storepass changeit -file server.cer -keystore server.keystore
```

Generate the client keystore and save in client.keystore

```
keytool -genkey -alias tomcat-cl -dname "CN=Client, OU=X, O=Y, L=Z, S=XY, C=YZ" -keyalg RSA -keypass changeit -storepass changeit -keystore client.keystore
```

Export the certificate from client.keystore client.cer

```
keytool -export -alias tomcat-cl -storepass changeit -file client.cer -keystore client.keystore
```

Import client's certificate(client.cer) into server's keystore

```
keytool -import -v -trustcacerts -alias tomcat -file server.cer -keystore client.keystore -keypass changeit -storepass changeit
```

Import server's certificate(server.cer) into client's keystore

```
keytool -import -v -trustcacerts -alias tomcat -file client.cer -keystore server.keystore -keypass changeit -storepass changeit
```

Figure 3 – Sample Message Level Encryption(WSE): Client Request

```

<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing" xmlns:wss="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
  <soap:Header>
    <wsa:Action wsu:Id="Id-3cc7e4b2-0a40-4b76-919a-dc1ee41198a4">http://stockservice.contoso.com/wse/samples/2003/06/StockQuoteRequest</wsa:Action>
    <wsa:MessageID wsu:Id="Id-399b6759-56dd-4190-bceb-7b515c902928">uuid:8e9e1467-2596-476e-9b2a-933f1ae3c5be</wsa:MessageID>
    <wsa:ReplyTo wsu:Id="Id-3ba798aa-615f-484a-b78a-6900d0833da6">
      <wsa:Address>http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous</wsa:Address>
    </wsa:ReplyTo>
    <wsa:To wsu:Id="Id-11f0cc21-908e-499d-a59d-3732f455efd1">http://localhost/ResponseEncryptCodeService/ResponseEncryptService.aspx</wsa:To>
  </wsse:Security soap:mustUnderstand="1">
  <wsu:Timestamp wsu:Id="Timestamp-6b1fc72f-6acc-41fb-828b-56b913ca0a18">
    <wsu:Created>2005-03-05T20:50:09Z</wsu:Created>

```

```

<wsu:Expires>2005-03-05T20:55:09Z</wsu:Expires>
</wsu:Timestamp>
<wsse:BinarySecurityToken ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-
profile-1.0#X509v3" EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-
security-1.0#Base64Binary" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" wsu:Id="SecurityToken-98186c7d-1b3d-4bce-b260-
1c978b03e65a">MIIBxDCCA W6gAwIBAgIQxUSXFzWJYYtOZnm muOMKkjANBgqhkiG9w0BAQQFAD
AWMRQwEgYDVQQDEwtSb290IEFnZW5jeTAeFw0wMzA3MDgxODQ3NTlaFw0zOTEyMzEyMzU5NTla
MB8xHTAbBgNVBAMTFFdTRTJRdWlja1N0YXJ0Q2xpZW50MIGfMA0GCSqGSIb3DQEBAQUAA4GN
ADCBiQKBgQC+L6aB9x928noY4+0QBsxnxkQE4quJI7c3PUPdVu7k9A02hRG481XIfWhrDY5i7OEB7K
GW7qfJotLLeMec/UkKUwCgv3VvJrs2nE9xO3SSWIdNzADukYh+Cxt+FUU6tUkDeqq7dqwivOXhuOTRy
OI3HqbWTbumaLdc8jufz2LhaQIDAQABo0swSTBHBgNVHQEEQDA+gBAS5AktBh0dTwCNYSHcFmRjo
RgwFjEUMBIGA1UEAxMLUm9vdCBBZ2ZVuY3mCEAY3bACqAGSKEc+41KpcNfQwDQYJKoZIhvcNA
QEEBQADQQAflbnMPVYkNNfX1tG1F+qfLhHwJdfDUZuPyRPucWF5qkh6sSdWVBY5sT/txBnVJGziyO8
DPYdu2fPMER8ajJfl</wsse:BinarySecurityToken>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo>
<ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#" />
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
<Reference URI="#Id-3cc7e4b2-0a40-4b76-919a-dc1ee41198a4">
<Transforms>
<Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
</Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<DigestValue>9vi/6TYy5E2JZGdymG5LHf04Eg=</DigestValue>
</Reference>
<Reference URI="#Id-399b6759-56dd-4190-bceb-7b515c902928">
<Transforms>
<Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
</Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<DigestValue>WsTSbJkEIUqqAEMDv0fmmlozgoY=</DigestValue>
</Reference>
<Reference URI="#Id-3ba798aa-615f-484a-b78a-6900d0833da6">
<Transforms>
<Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
</Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<DigestValue>6fV4FHAE1j1vcuQyk7yMFoJFUS8=</DigestValue>
</Reference>
<Reference URI="#Id-11f0cc21-908e-499d-a59d-3732f455efd1">
<Transforms>
<Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
</Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<DigestValue>GJhbB8kFWR1B3SQjRcz6UC4ZdTk=</DigestValue>
</Reference>
<Reference URI="#Timestamp-6b1fc72f-6acc-41fb-828b-56b913ca0a18">
<Transforms>
<Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
</Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<DigestValue>d50wqaew1ERJN0HODXFHI19WD0=</DigestValue>
</Reference>
<Reference URI="#Id-552ffa38-a006-4ba1-a49a-50cf382160ad">
<Transforms>

```

```

<Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
  </Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<DigestValue>6FjliOnYyisrtRp10DQghULchdo=</DigestValue>
  </Reference>
</SignedInfo>

  <SignatureValue>LAdQiF3LdMF1Ic30GGsw5YIyUGHTq7ICWmNgx/abKzgZRZSgKXY9U2yd7CggcBjYf
  WdMVXSw6xISfVwkj4eO8IB1QSWEqWmNklP5j4MIUjHeWiqww4mEzEejC7A88JMdbqD7/duZLrIc5MHE
  ezXfQ2ar/S18VkZ1sd7rcNkWaGE=</SignatureValue>
- <KeyInfo>
- <wsse:SecurityTokenReference>
  <wsse:Reference URI="#SecurityToken-98186c7d-1b3d-4bce-b260-1c978b03e65a" ValueType="http://docs.oasis-
  open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" />
  </wsse:SecurityTokenReference>
  </KeyInfo>
  </Signature>
  </wsse:Security>
  </soap:Header>
- <soap:Body wsu:Id="Id-552ffa38-a006-4ba1-a49a-50cf382160ad">
- <StockQuoteRequest xmlns="http://stockservice.contoso.com/wse/samples/2003/06">
- <symbols>
  <Symbol>test</Symbol>
  </symbols>
  </StockQuoteRequest>
  </soap:Body>
  </soap:Envelope>

```

Figure 4 - Sample Message Level Encryption(WSE): Server Response

```

<?xml version="1.0" encoding="utf-8" ?>
- <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing" xmlns:wsse="http://docs.oasis-
  open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-
  open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
- <soap:Header>
  <wsa:Action>http://stockservice.contoso.com/wse/samples/2003/06/StockQuoteRequestResponse</wsa:Action>
  <wsa:MessageID>uuid:4e83fe61-ba1d-4b7a-88d0-9b7866e881bf</wsa:MessageID>
  <wsa:RelatesTo>uuid:8e9e1467-2596-476e-9b2a-933f1ae3c5be</wsa:RelatesTo>
  <wsa:To>http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous</wsa:To>
- <wsse:Security soap:mustUnderstand="1">
- <wsu:Timestamp wsu:Id="Timestamp-697f2f57-679d-4fee-90be-2cfd9c823fda">
  <wsu:Created>2005-03-05T20:50:30Z</wsu:Created>
  <wsu:Expires>2005-03-05T20:55:30Z</wsu:Expires>
  </wsu:Timestamp>
- <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmenc#">
  <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmenc#rsa-1_5" />
- <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
- <wsse:SecurityTokenReference>
  <wsse:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
  1.0#X509SubjectKeyIdentifier">gBfo0147IM6cKnTbbMSuMVvmFY4=</wsse:KeyIdentifier>
  </wsse:SecurityTokenReference>
  </KeyInfo>
- <xenc:CipherData>

```



```

    <xenc:CipherValue>hN2tWQE4J4nXEEjOBy/371jeTTARr7tdnd+JhD/y67UTv537JKR+BGyWoQ3LezUm
    MEf6fZOprvhn+0vCSNSj4jLiiWV+L6AHujiBRv7Zsc6Agz2Fs9eyX+ugojXS3C0Xgr9z+PVGiW7+Yysm1/9
    ES2MF335qbGHZy2ZrAf3w=</xenc:CipherValue>
  </xenc:CipherData>
</xenc:ReferenceList>
<xenc:DataReference URI="#EncryptedContent-f4439e22-1ab9-446d-9a71-0a9a1d3363ce" />
</xenc:ReferenceList>
</xenc:EncryptedKey>
</wsse:Security>
</soap:Header>
<soap:Body>
<xenc:EncryptedData Id="EncryptedContent-f4439e22-1ab9-446d-9a71-0a9a1d3363ce"
  Type="http://www.w3.org/2001/04/xmlenc#Content" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
  <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc" />
</xenc:EncryptedData>

  <xenc:CipherValue>hbn2rTnaZx8AhQBSkZWXH5lB+Dcnu4bm8mcUfHAZtfZCbKjF6TbsiPk7FcaOG92EoG
  EFFitxd6BSLZD5G+McpQ7409jA8xr3qXhkM6Tdwr6BuBG/IEGC2dH3OyOgYhevtzdzJj1V3nEvIv4fCkk
  A5wflN/eRPLfks5wjFOZtMnRilEk2J2Q5ny4UpH9O0+NI0dBzhzhei3CYVuyZtElleDbWu2byh6ZzH3/u02hg
  LasFy8hQDYNSSd61YMm5tgyfDCauL3VW8DrGNuHGwPFBTg81z6LgxDji3ICTtPjGNNtXZF6LFEYNzd
  slMKMoaf4ZWuXMGk6d+z8VcosivXtwOrT83HBoAkstVdaSYivq62ET3DHRJYUqX8iiQvUoLqs0GQPDa
  ewPuZrEQdiVrhmb2hww8s7c61mhAC62DF4uGSpMYJgCR9ri2Tcbf/3WdWBVSU0UIa2mQyTcf5QOCfpcj
  Z0zXjdPyImITiLiAeeeZOkbD33+ncK/X/7rvFXRZ59/nNtZjrTCL4pR57ryGeOLP7nATkD53cheKPtekgOkE
  CGYu1sODw5BuFpYMIgpGs/D6lFrkNIHVvPFEkpRDu6MQ==</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedData>
</soap:Body>
</soap:Envelope>

```

© SANS Institute 2000 - 2005. All rights reserved.

X. LIST OF REFERENCES

- "Axis User Guide." Apache Project. 2005. 7 March 2005.
<<http://ws.apache.org/axis/java/user-guide.html>>.
- Clements, Tom. "Overview of SOAP." Sun Microsystems, Inc. 2002. 7 March 2005.
<<http://java.sun.com/developer/technicalArticles/xml/webservices/>>.
- He, Hao. "What is Service-Oriented Architecture." XML.com. 2003. 3 March 2005.
<<http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html>>.
- "Introduction to WSDL." W3Schools. 1999. 3 March 2005.
<http://www.w3schools.com/wSDL/wSDL_intro.asp>.
- Joung, Philip. "Web Services vs. You: Who's Winning?" Software Test & Performance. 3 September 2004.
- Mahmoud, Qusay. "Accessing and Interacting with Remote SOAP-enabled Services." Sun Microsystems, Inc. 2004. 3 March 2005.
<<http://java.sun.com/developer/technicalArticles/WebServices/SOAP/>>.
- "Secure Sockets Layer." Whatis.com. 2005. 7 March 2005.
<http://whatis.techtarget.com/definition/0,289893,sid9_gci343029,00.html>.
- Smith, Robert. "SOAP/XML Firewalls." WindowsITPro. 2003. 8 March 2005.
<<http://www.windowsitpro.com/Windows/Article/ArticleID/39755/39755.html>>.
- "Technical Overview of SAML." OASIS. 2004. 8 March 2005. <<http://www.oasis-open.org/committees/download.php/6837/sstc-saml-tech-overview-1.1-cd.pdf>>.
- Walsh, Norman. "A Technical Introduction to XML." XML.com. 1998. 3 March 2005.
<<http://www.xml.com/pub/a/98/10/guide0.html>>.
- "Web Service Architecture." World Wide Web Consortium. 2004. 3 March 2005
<<http://www.w3.org/TR/ws-arch/>>.
- "Web Services Enhancements." Microsoft. 2005. 7 March 2005.
<<http://msdn.microsoft.com/webservices/building/wse/default.aspx>>.
- "Web Services Security." OASIS. 2004. 8 March 2005. <<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>>.
- Wong, Wylie. "Why Web Services Make Business Sense." CNet News. 2001. 1 March 2005. <<http://news.com.com/2009-1017-275442.html?legacy=cnet>>.