

**Secure Authentication and Session**  
**State Management for Web Services**

Clay Lehman

CSC 499: Honors Thesis  
Supervised by: Dr. R. Michael Young

## 1. Introduction

Web services are a relatively new technology that allow a client to communicate with a server by sending messages using the HTTP protocol. In a web service, the server invokes a function based on the message sent from the client and then sends a response back to the client. Web services allow a range of new functionality to be added to web pages because the page can request a function to be run on the server and then change based on the results. Another way web services are helpful is in communication between applications. When a client communicates with the server using web services, the communication is language independent. This means that a client can be written in any programming language and communicate with the server without knowing what language the server's web services were implemented in. There are a number of potential uses for web services because they are versatile and implementation-independent. Any two applications can communicate together through a web service regardless of how they are implemented.

The Mimesis project is currently working to produce a game engine that can send requests to an Advisable Planner to control the AI (artificial intelligence) for the environment of the game. The Advisable Planner will use a web service named Fletcher to accept requests from client game engines for a new plan. A game engine will communicate with the Fletcher web service, request a plan, and then execute the plan in the engine locally. A web service is an effective way for the communication between the game engine and the Advisable Planner because it can communicate over the internet, and is language independent.

Web services will provide a way for a game engine to ask a planner on a central server for a plan. Since web services are based on HTTP which is a stateless communication protocol, the problem of maintaining the history of a client's transactions can be difficult to solve. Furthermore, a server may only want to provide services for registered users, but there is no built-in way to handle authenticating users who call a web service. The Advisable Planner needs to be able to store information about the history of a client's calls, so it needs to use authentication to verify who the user is, and also needs a way to store important data about the client's previous transactions. There is little work focusing on the interaction between a web service and an individual client, so this problem has not been fully explored by previous research. This paper will suggest how an application can approach state management and user authentication for a web service application to allow a web service to deal with each client individually and maintain information about each client's previous transactions.

## **2. Previous Work**

Using web services for complex tasks that require more than one communication is a relatively new idea. Ardissona, Goy, and Petrone [2] researched using web services for complex tasks that require more than two messages to be passed. There was no real accepted protocol for dealing with a complex web service that required a set of transactions to perform a task, so Ardissona, Goy, and Petrone introduced a method of formalizing a conversation between a web service consumer (client) and a web service provider (server). The ideas behind this protocol were based loosely on the Speech Act theoretical model of a conversation. The Speech Act Theory uses the participants'

communication actions (Speech Acts) to define their roles (speaker/hearer) in the conversation. These roles then define the order of their turns in the conversation. This decides which participant should be speaking and then when to respond. Ardissona, Goy, and Petrone applied this theory to introduce a lightweight way to manage conversations between providers and consumers of web services. They went on to provide the framework for server management of these interactions. This research is helpful to a complex application that needs multiple calls to a web service provider to complete a task, and the chain of messages can maintain some data that is necessary for the task, but in the case of the Advisable Planner, there is a need for more long-term data storage for each individual user.

Security is a major topic in the computer world, and there has been a lot of research into schemes for encryption, hashing algorithms, and ways to prevent different kinds of attacks. As web services became popular, a major problem is keeping these applications secure so that they can be used for e-commerce without giving away customer information. Other applications using web services also need security to prevent their data from being viewed or stolen by attackers. Microsoft researched ways to secure web service applications and introduced a standard for web service security, calling it WS-Security [3]. WS-Security addresses all aspects of secure communication with web services, everything from ways to use tokens to specify who a user is, to signing messages that are sent, and even schemes for public/private key encryption of messages. WS-Security lays out the minimum requirements for each of the security elements it addresses in order for a web service to be considered secure by their standards.

Bhargavan, Fournet, and Gordon [5] looked into some of the methods for security suggested by WS-Security and devised a way to prove the security of an algorithm based on formal representations, and pi calculus. The security protocols these researchers explored were: password digest authentication, password-based signatures, X.509 signature, and firewall-based authentication. Password digest authentication is when the user sends an authentication string that includes the username, a digest (hash) of the password, and a timestamp. The server then finds the user's information and compares the password digest to a digest of the correct password. The timestamp must be recent or the server will reject the user's login. Password-based signatures require the user to sign the envelope of the SOAP message sent to the server with a signature based on their username and password. X.509 signatures use public-key signatures to sign their messages so the server can be sure the contents of the message have not been viewed. This way signatures are not based on the user's authentication credentials. Finally firewall-based authentication is a way to add flexibility to a web service. For example if a web service allowed either password signing or X.509 signing, there could be a firewall that has the database of passwords. This firewall will accept either type of signature, check its validity, then add the other type of signature, and a firewall header that says the message has been authenticated and then passes the message to the appropriate server [5]. The server will check whichever type of authentication it requires, but both signatures have been included by the firewall. Bhargavan, Fournet, and Gordon go on to use pi calculus to formally represent password-based authentication with signatures and prove that this method is secure enough for "robust safety" for the messages passed.

Damiani, Vimercati, and Samarati [8] explore two of the popular approaches to web service security that have been proposed. The first method is to encrypt data in the XML document sent between the server and client with signatures. This method would allow flexibility because parts of the XML document's tree could be encrypted while others are left plain text. A message could also include different parts encrypted with different keys so only certain users could read certain parts of the message. This is a good approach to securing general XML documents, but SOAP messages are XML documents with a specific structure and could benefit more from a specific security scheme designed for SOAP messages. The second method Damiani, Vimercati, and Samarati explored is using SOAP headers to pass credentials and authentication information. SOAP messages are composed of a body and a header. This header can be customized to include authentication information. SOAP headers can host digital signatures, usernames, passwords, and any other type of authentication information chosen by a developer. Using SOAP headers for credential transfer is even one of the ideas presented by Microsoft's WS-Security. Damiani, Vimercati, and Samarati came to the conclusion that research in web services security is making good progress and that using SOAP headers for credentials is a good step in the right direction.

### **3. Background**

A web service is a set of protocols to allow communication between two applications. A client and a server can pass messages back and forth to exchange data over the internet similar to how two processes communicate on a single computer. There are several ways to pass these messages, but the most common is HTTP [16]. Web

services are “document oriented” rather than object oriented. This means that web services are described by simple XML documents.

Web services are invoked by sending Simple Object Access Protocol (SOAP) messages over the internet. SOAP messages are XML-based documents. The client sends a SOAP request to the server and then the web service performs some task and sends a SOAP response back to the client. A SOAP message consists of a header and a body. The header contains metadata about the message. The body contains the actual message, in a request sent to the server, the body specifies a web service to be invoked and any parameters and return variables needed to call a web service, in the response from the server the SOAP body includes the variables returned by the web service [12].

Web Service Description Language (WSDL) is an extension of XML that is used to define a web service. Every function, parameter, and return type made accessible by the web service is defined in the WSDL file. Using WSDL allows web services to be defined while still being platform independent. WSDL are a key element in the Universal Description, Discovery and Integration (UDDI) Registry which Microsoft provides. A business can publish the WSDL file for their web services on the UDDI Registry and then a user can discover their web services and create client application to access the web services. Since the web services are described by platform independent WSDL files, the client application can be written in any language and communicate with the web service seamlessly. The ease of discovery and platform flexibility make web services a very powerful tool, and have led to wide usage in all types of applications where two modules written in different languages need to communicate over the internet to accomplish a task.

In the Mimesis project, the Fletcher web service will allow a game engine to request a plan and then communicate with the Advisable Planner to generate and return an appropriate plan to the engine. In many cases the engine will request a plan, alter the options, and request a new plan based on these options. The web service needs to remember what the last plan was in order to help the Advisable Planner keep track of what state it was in when the client last requested a plan. The web service will also need to restrict access to registered users so that it can maintain state based on previous transactions with each user individually. This means that Fletcher will need to handle state management and user authentication in its web service to effectively create plans for the client game engine.

Since web service communications are document-based instead of object-based, there is generally no way for a web service to retain information about what users have requested in the past. HTTP is a stateless protocol, so SOAP messages do not provide information on whether a sequence of requests are all from the same user, or all from different users. This type of information is not needed for simple web applications where a client just requests some piece of data and is done. However, in a complex system with two modules communicating exclusively through web services, it is sometimes necessary for the web service to remember information about the system's state after the last call by a user. Fletcher will need to maintain this kind of state information for a session between transactions for the Advisable Planner. Since this functionality is not provided by WSDL web services, the application has to find a way to maintain this state information during a communication session between the client and the web service. Depending on the development environment used to create and publish the web service,

some basic methods of maintaining session state are generally provided. ASP .NET is Microsoft's development environment for web services, and along with a compatible web server, ASP .NET provides a session object that can store data to be retained between calls to a web service. This feature is helpful when creating complex applications that need to save session state data. In this scheme, the client application is responsible for maintaining identification information about its session so the server knows it has not ended the session. The problem with this approach is that the server has no idea which client is using the web service, or even if the client is a registered user that should be allowed to access the web service.

In any situation where information is going to be passed over the internet, security is a significant issue. Encryption, key sharing, and user authentication are all popular research areas in the computer science field today. Encryption is the process of encoding a message so that only the sender and receiver are able to decode and read the contents of it. One way of encoding messages is to use a key. When you use a key for encryption, the sender and receiver both have to know the secret key. This brings up the issue of sharing this key without allowing others to find it out. User authentication is the process of authenticating a user based on a username and password before allowing the user access to a system. When developing a protocol for user authentication, one has to make sure that a user's information is not compromised when logging into the system. This means the system needs some form of encryption built into a login system to prevent passwords from being compromised.

Complex web services may need to identify a user who is making a request to the server. The server may want to restrict access to the service only to registered users, or

even provide specialized interactions based on a user's information and previous transactions. Fletcher will need to restrict access to only registered users, as well as provide specialized service for each user. In this situation, the web service needs to find a way to handle user authentication. While web services are becoming an increasingly popular way to allow communication between applications over the internet, there is nothing in the underlying technologies of web services that even addresses user authentication. Since web services have become more complex and are being used in new ways, web service security is a major issue in computer science research. Since the specifications for web services do not address user authentication, it must be implemented by the developer of an individual web service [8]. Creating a method of authenticating users of a web service is not a trivial task, and the developers must pass user information carefully, to ensure no one intercepts the passwords as they are sent to the server.

A database system will be helpful in the web service for Fletcher because it will have to maintain state information, as well as usernames and passwords for all of its registered users. One common database implementation is a Structured Query Language (SQL) database. SQL is a language standard that is used to create, modify, and access information in a database. SQL statements allow a user to easily and efficiently add information to a database and find information in a database [13]. This is the common way for systems save lists of usernames and passwords for login information.

#### 4. Research

Security and authentication are a necessity in the Fletcher web service, because the system will need to store session data for each individual user. To prevent storing data for users who are not registered to use the system, secure authentication and authorization is needed. Since SOAP messages and WSDL specifications have no mention of security, this is a difficult problem to solve. Microsoft's work on WS-Security has many specifications for ways of creating secure web services that are helpful when creating an authentication schema for a service. WS-Security addresses the three main categories of security: authentication, integrity, and confidentiality [1]. Authentication is preventing unauthorized users from accessing published web services. Integrity is ensuring that the data received was not altered while being transferred over the internet. Confidentiality is making sure that messages sent by a user cannot be read by anyone other than the server.

To address authentication, WS-Security suggests including a username token in the SOAP header which contains a username, hash of the password, timestamp, and nonce. The username will be how the user is registered with the server and sent in plain text. WS-Security requires the password to be hashed using the SHA-1 algorithm [1]. Since SOAP messages are sent over networks in plain text, the password must be hashed to prevent malicious unregistered users from discovering passwords and gaining access. A timestamp tells the server when a message was sent, and the message times out if the timestamp is too old. A nonce is a random, unique string generated by the user. The combination of the timestamp and nonce prevent replay attacks by unregistered users.

WS-Security requires a signature in SOAP messages to prove message integrity. A user is required to use the XML Signature specification provided by W3C to send a signature for each message sent to a web service. The signature is defined using a <Signature> tag in the SOAP message and included in parts of the security header. The signature should not be wrapped around the entire header, as required in the XML Signature Specification [3]. Instead, WS-Security suggests signing certain sections of the message. This way a message can be signed in different sections with multiple signatures, allowing one message to flow through different stages in a distributed application. If a web service receives a request with an invalid signature, the message is rejected because it could have been tampered with as it passed over the network.

WS-Security also specifies a method for providing confidentiality in a web service. WS-Security suggests following the XML Encryption specification provided by W3C for encrypting confidential information sent in SOAP messages [3]. The encrypted key should either be a shared key by the sender and receiver, or a key that is included in the message itself. Encryption should be used only in portions of a message, similar to how signatures are treated.

The ideas presented by WS-Security are very solid and helpful for creating a web service application that is robust and secure. However, in the Fletcher web service, some of these precautions may be unnecessary. Fletcher needs secure authentication to prevent unauthorized access, and to identify users for its session information, but the information returned by Fletcher does not necessarily need to be signed and encrypted because it is not confidential information.

The best solution for Fletcher is some sort of simple authentication that the client can include in a header which encodes a string that includes the password and a timestamp using the SHA-1 hashing algorithm. SOAP messages are made up of a body and a header, and you can extend a SoapHeader class to create a custom header with fields for a customer to send username and password information. Adding a timestamp to the password before sending the message will prevent replay messages because the hashed password will change regularly [4]. A timestamp that is precise to the minute should prevent most replay attacks, but the server needs to deal with timestamps which were sent very recently but may be a minute or two off. Providing a web service to return the current time on the server will prevent the problem of synchronizing system clocks. However, if the client requests a timestamp at hh:mm:59 then the timestamp will most likely change before the client's message gets back to the server to request a secure service. This can be prevented if the server checks the client's password and timestamp digest against timestamps from the last several minutes, depending on how long a timestamp is decided to be valid for. Another option would be to use a nonce instead of a timestamp, but then you would have to find a way to have the server and client generate the same nonce, but one that a malicious user could not intercept. This simple but secure approach to authentication would best suite Fletcher because it keeps authentication information out of the parameters, and a simple call to a verification method can query the database to verify the user's password. This approach keeps the web service code from becoming overly complicate due to authentication. This also will provide a lightweight solution so the efficiency of calls to Fletcher should not be unacceptably slower due to the addition of user authentication.

Once authentication has been implemented for Fletcher, the server knows which user is making a request for a new plan. At this point, Fletcher needs to be able to maintain information about the state of the previous session with this user. This information will help the Advisable Planner to create a new plan that is unique, rather than a plan it has already sent the client earlier. For this, Fletcher has to have a way of maintaining session state for a user between calls. This session state will include information about previous transactions, and options the user has already input for the planner. While WSDL does not have any built in functionality for state management, there are some options for ways that one can maintain state information in web services. One way to maintain state information is to create a global Singleton object that the web service can access when it is called. A Singleton object is a common programming pattern where the object can be accessed by other processes, but only one copy of the object can be created. In the ASP .NET environment, there is a way to maintain session information in a session object that the server maintains. Another option is to save session information in a SQL database with the user as the key of the database.

The Singleton pattern is a common software pattern where a class is globally accessible, but only one object from the class can be created. The constructor has to be declared private, and a public static method is provided to obtain an instance of the object. If there is not current instance of the object a new one is made, otherwise the pointer to the single object is returned [7]. For state management, a web service could use a singleton object to keep pointers to a session object for each user who has recently communicated with the server. This would allow the web service to save information in the session object, and access it later through the public static singleton object. This

would work for the Fletcher project's needs, but there are other options that should be explored.

The ASP .NET environment provides session memory that can be accessed to maintain state information for web services. This would be a good solution for Fletcher's state management needs. The web service can save information about the state of the transaction for each user, and the information will be accessible for all of the user's requests. This session object is kept between calls, but the user must provide the session identifier to keep the session open [11]. In this case, when the client application is closed, the session identifier is lost, and therefore the session information is lost. This is a problem for Fletcher because a way to maintain long term session information is also needed.

Another possible solution for maintaining state information for each user is to save the important information to a SQL database between requests to the web server. The web service could have a table in a database that stores certain data on the SQL server for each user, using the user as the primary key. This would allow for long term storage of information rather than losing the data after each session the user has. The problem with this is that it is expensive to access and query the database query every time a call is made to the web service.

The best solution for Fletcher is a combination of the session state memory provided by ASP .NET for short term information, and a SQL database for storing information between sessions. The web service can store all the necessary information in a session object and access it as a client makes a sequence of requests. If a user ends a session, or if a session is timed out, the web service will back up the important session

information to the SQL database so that it can be accessed the next time the user connects to Fletcher. Since users will be required to provide authentication information to connect to Fletcher, saving information to the database by username will be easy because the server will know exactly who the session belongs to.

## 5. Example

### 5.1 Authentication

To handle authentication, there are three main steps that need to be addressed. The first problem is deciding how the client provides authentication information to the server. The second step is to ensure that the information is passed in a secure manner so observers cannot acquire a user's authentication information. Finally the server has to have a way to verify that a user's information is correct before they are given access to a session.

SOAP messages are made up of a header and a body section. This makes them a good method of passing authentication information, because the developer can add username and password fields to the SOAP header. In ASP .NET a SOAP header can be extended to create a custom SOAP header class that includes a user name and password. Creating the custom SOAP header in the server's web service code allows the client to set this information to be sent to the web service. The following code creates this custom SOAP header class:

```
Public class AuthHeader : Soapheader
{
    public string Username;
    public string Password;
}
```

The client needs access to this class in the web service code, so the following code will create an instance of the web service (named “Service1”) and set the user credentials:

```
Service1 service = new Service1();

AuthHeader Auth = new AuthHeader();
Auth.Username="name";
Auth.Password="pass";

Service.AuthHeaderValue=Auth;
```

This assumes that there is a global variable named “AuthHeaderValue” in the Service1 class. The client can now invoke the web service that required an AuthHeader to be accessed.

To require authentication credentials in a web service, the server must use specific attributes when declaring the WebMethod. The following code will declare the method authenticateUser() to be called by a client:

```
[SoapHeader ("Auth",Required=true)]
[WebMethod()]
public string authenticateUser()
{
    ...
}
```

Now the server has required an authentication header and the client has provided one. The next step is to make sure this information is not intercepted by observers. The following code obtains a Timestamp from the server and then uses SHA-1 to encrypt the password and the timestamp into a single string to be sent to the server:

```

public string passwordDigest(string pass)
{
    DateTime dt = service.getTimestamp();

    //this concatenates the timestamp, accurate to the minute
    //with the password
    String hpass=pass+dt.ToString("yyyyMMdd")+dt.ToString("HHmm");

    //this uses SHA1 to hash the password+timestamp string
    Return BitConverter.ToString(
        new SHA1CryptoServiceProvider().ComputeHash(
            Encoding.Default.GetBytes(pass))).Replace(
            "-",String.Empty);
}

```

The server now has to compare the hashed password string to a hashed string of the user's correct password. If these two string match, then the user is authenticated. To store registered users data, a server should use a database. The following code will look up a user in a MySQL database:

```

private int verifyUser(string user, string pass)
{
    MySqlConnection conn= new MySqlConnection(
        "server=www.server.com;database=db_users;uid=name;pwd=pass");

    string query = "select password from users where
                    username=\'"+user+\'";
    MySqlCommand command = new MySqlCommand(query,conn);
    string hashedpass=(string)command.ExecuteScalar();
}

```

With this information, the server can simply compare the "hashedpass" to the password sent from the client and if they are the same, the server can be sure the user is allowed to access the system.

## 5.2 Session State Data

Session state data can be saved automatically in the ASP .NET environment. To allow this functionality, the method needs to be given special attributes when it is declared. The following code declares a web service method that will retain session data between calls from a client:

```
[WebMethod(EnableSession=true)]  
public string rememberMe()  
{  
    . . .  
}
```

To save a variable in a method that has sessions enabled, you simply assign values to variables and store them in the Session object:

```
string date = "3-28-2005";  
Context.Session["sessionDate"]=date;
```

The job of keeping a session ID is left up to the client. If the client is a web page, then the cookies are automatically the same, unless the browser is closed and reopened. If the client is an application, the following line needs to be included in the constructor to make the application maintain cookie information:

```
this.CookieContainer=new CookieContainer();
```

The built in session object is a good method of maintaining short term session data, but some applications, like Fletcher require some data to be maintained over a longer period. This can be achieved by saving the important information to a database when a user ends a session, or when the session times out. SQL calls will be similar to the previous example for retrieving a user's password, however the developer will have to design the database based on what information they wish to store, and generate query strings to properly save this information to the database. The string to store the

“sessionDate” variable used in the session example assuming a row for the user was already inserted, but the “sessionDate” value was NULL would be the following:

```
string query = "UPDATE \'users\' SET \'password\' WHERE  
              \'username\' =\'"+user+"\'";
```

The examples provided here have demonstrated how to provide secure authentication and session state management in a web service using the ASP .NET environment. The possibilities with using web services are endless, and more and more businesses are turning to them for web applications every day. With support for authentication and session management, web services will have even more room to grow.

## 7. Conclusion

Web services are a rapidly growing technology, especially in the e-commerce area. Web services have a lot to offer when it comes to creating web-based applications for selling things over the internet. They are also a good way for applications to communicate with each other over the internet. This allows applications implemented in different languages to cooperate seamlessly in a larger system. This makes web services a good option for the Mimesis project. A game engine on a users computer needs to request a plan from a centrally located Advisable Planner to decide what actions the engine will take. These two systems are written in different languages and must communicate over the internet. The Fletcher web service will provide this communication link between the game engine and the Advisable Planner. The solutions discussed in this paper will allow Fletcher to provide secure authentication and state management for this communication, which is required for the Advisable Planner to properly generate a series of plans for the game engine.

References:

- [1] “An Introduction to Web Service Security Using WSE – Part I.”  
<http://www.codeproject.com/webservices/WS-Security.asp>.
- [2] Ardissono, L., Goy, A., and Petrone, G. “Enabling Conversations with Web Services.”
- [3] Atkinson, B. “Web Services Security (WS-Security).”  
[msdn.microsoft.com/blabla](http://msdn.microsoft.com/blabla)
- [4] “Authentication for Web Services.”  
<http://www.codeproject.com/cs/webservices/authforwebservices.asp>.
- [5] Bhargavan, K., Fournet, C., and Gordon, A. “A Semantics for Web Services Authentication.”
- [6] “Build Secure Web Services with SOAP Headers and Extensions.”  
[http://www.codeguru.com/Csharp/Csharp/cs\\_webservices/security/article.php/c5479](http://www.codeguru.com/Csharp/Csharp/cs_webservices/security/article.php/c5479).
- [7] “Creating Singleton Objects using Visual C++.”  
<http://www.codeproject.com/gen/design/singleton.asp>.
- [8] Damiani, E., Vimercati, S., Samarati, P. “Towards Securing XML Web Services.”
- [9] “Nine Options for Managing Persistent User State in Your ASP .NET Application.”  
<http://msdn.microsoft.com/msdnmag/issues/03/04/ASPNETUserState/default.aspx>.
- [10] “Session Based Singleton Object.”  
[http://aspalliance.com/articleViewer.aspx?aId=70&pId=.](http://aspalliance.com/articleViewer.aspx?aId=70&pId=)
- [11] “Session State.” <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconSessionState.asp>.
- [12] “SOAP Tutorial.” <http://www.w3schools.com/soap/default.asp>.
- [13] “SQL Tutorial.” <http://www.w3schools.com/sql/default.asp>.
- [14] “Using ASP .NET Session State in a Web Service.”  
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnservice/html/service08062002.asp>.
- [15] “Web Service Authentication.”  
<http://www.codeproject.com/useritems/WebServiceAuthentication.asp>.
- [16] “Web Services Description Language.” W3C.  
[http://www.w3.org/TR/wsdl#\\_introduction](http://www.w3.org/TR/wsdl#_introduction).