Secure Common Web Server Session

Sharing object data across deployed Java web applications on the same web server

Chad Cook and Lei Chen

Department of Computer Science, Sam Houston State University, Huntsville, TX 77341, USA

Abstract - *When web applications are deployed to a Java web server, there is no consistent or easy way to share object data among them. In this paper, we propose a mechanism, the Secure Common Web Server Session (SCWSS), which allows object data to be shared across deployed web applications, independent of the web server or any other implementation specifics, in a manner similar to storing session objects in Java. In SCWSS, the byte representation of the object data is first encoded to ASCII format, then encrypted (currently using DES), and finally saved in a cookie with a name supplied by the developer at the root level. Data can then be retrieved by any other application deployed to the same web server that can supply the correct encryption key. The proposed mechanism has been implemented, tested using various browsers, and analyzed for shortcomings and possible improvement.*

Keywords: Secure, web Applications, Java, session, cookies, encryption

1 Introduction

When developing Java web applications, there is often a need to store data, objects, for use elsewhere in the application. This can be conventionally done by saving data for that specific user's session which can then be retrieved elsewhere in the application. While useful for a single web application, it cannot be used to share data across web applications deployed to the same web server. Some web application vendors offer settings and mechanisms for allowing session data to be shared across web application, but these require administrator level access and developers may not have access to these permissions.

One method that is available for sharing data across web applications is by storing a cookie in the client's browser. However, this method only stores string data, which is not very useful to an object-oriented language such as Java. This paper proposes a method of using this cookie mechanism to share data in a secure manner between web applications by converting the object data into a string representation of that object, encrypting it, then storing it in a cookie for retrieval by any web application that has the proper key to decrypt it, and converting the string back into the original object.

The rest of the paper is structured as follows. Next in Section 2, we discuss the advantages and disadvantages in the existing ways to share data across web applications. We outline a number of key terms and concepts used throughout the paper in Section 3. An overview of the Data Encryption Standard (DES) scheme, the encryption standard we used to share data across web applications in a secure way, is introduced in Section 4. This is followed by description of the detailed implementation in Section 5, which describes object data sharing, implementation environment, diagrams of Java classes developed, and the code flows of object storage and retrieval processes. The implementation environment is described in Section 6 and is tested and analyzed in Section 7. At the end of the paper we discuss some of the known shortcomings of our implementation in Section 8 and areas for future enhancements and improvements in Section 9.

2 Background

Every now and then web application developers need to store data or objects to be used later in applications. Therefore session data need to be store in a reliable and sometimes secure way to preserve data confidentiality and integrity. When only a single web application is involved, the above method may be quite competent. However in a multiple web applications environment, there exist a number of potential problems, the most noticeable problem being the settings and mechanisms for allowing such session data sharing require administrator level access which developers and web applications normally do not have.

Currently, when sharing data between web applications deployed to the same web server, there are a couple of ways to do so. Some vendors offer settings at the server level to provide ways to share session data (also known as context sharing). For example, when using WebSphere Application Server V5, the WebSphere extension to the servlet 2.3 API allows sharing session context across an enterprise application [14] [15]. Session attributes must be serializable to be processed across multiple Java Virtual Machines (JVMs) [11]. Reliability and availability of a user's session state must be guaranteed. The In Process and Out of Process methods used by ASP.NET can be configured to maintain session state [10] in a reliable way. While this can be useful, it does not work if the applications need to be deployed to a

different vendor's web server that does not offer context sharing or if the developer does not have access or a way to modify the server's settings to enable context sharing.

Data has also been shared by writing its binary counterpart to a database, where other applications can then access it. While this approach is web server vendor independent and the developer does not necessarily have administrator access, it is a very tedious approach, requiring changes to the database whenever new objects need to be saved and possible modification to existing database metadata when changes to the objects occur. When the database environment changes, all tables must be copied over, which can be tedious if this is a switch between vendors.

Another method that allows data to be shared is by using a cookie [5] [8]. A cookie is string data that is stored in the client's browser and resides at the level of the web application that creates it. However, the path of the cookie can be created to the root level, giving all web applications deployed to the server access to it. While this approach does not require any administrative changes and is vendor independent, the fact that it uses cookie data limits the data that can be shared to strings. But if there is a way to convert object data to a string and vice-versa, a simple, vendor independent approach that requires no administrative assistance becomes available to the developer. Our goal was to develop such a solution. In the next section, we introduce a number of terms frequently used in our secure object data sharing mechanism.

3 Key terms and concepts

Below are a number of the key terms and concepts used in this paper:

- Client: the end user, typically referring to the end user's computer or computing device. The term client also refers to client side applications. The client sends requests for data and/or to perform actions to a server.
- Server: the machine where the web server and web applications deployed to it are running. The term server also refers to the server applications. When the server receives a request from a client, it sends a response after processing the request.
- Web Server: a piece of software that is run on a machine with an internet connection. It processes requests for web pages and other internet related data and typically sends data back after processing the request.
- Web Application: an application which is based in the web browser. A single web application can be run for many different users at once, each with their own session.
- Cookies: string data which is saved on the client's machine. This is typically informational data and can be included if a user has accessed this web application before and any user (client) preferences.
- Data Encryption Standard (DES): a widely used encryption standard. DES is covered in detail in section IV.
- Java Session: when a client first accesses a web application on a web server, a unique session is created for that specific web application. This session "lives" while the client interacts with the web application. They can store and retrieve information in the session while they interact with the web application. However, they cannot share data with other web applications and once the client terminates their interaction with the web application the session is removed.

4 Data Encryption Standard (DES)

The Data Encryption Standard Scheme (DES) [3] [4] is a standard encryption scheme, used both by the government and privately [12]. DES is a symmetrical encryption algorithm as it uses the same 64-bit key (56 bits for encryption, 8 bits for parity checking) to both encrypt and decrypt the data. This is done by providing a key to encrypt the plaintext data, resulting in encrypted text, or ciphertext. To decrypt the ciphertext, the same key used to encrypt the data must be supplied to decrypt the data back to its plaintext form.

To perform the actual encryption, data is split into 64 bit blocks and then fed through 16 rounds of processing. To perform each round of processing, the 64 bit block of text is split into two 32-bit halves. Each half is then expanded by using substitutions and permutations of bitwise shifts and reordering, resulting in 48 bits. Then a subset of the key is combined with the 48 bits using an XOR operation and the block is again divided into smaller pieces where each piece is fed into a substitution box where a non-linear transformation is used to ensure that the cipher will not be trivially breakable. To perform the decryption, the process is run in reverse order.

Since a brute force attack against encryption using a 56 bit key is relatively easy to perform [6], the DES process can be repeated two more times, called Triple DES and in effect creates an encryption key of 56, 112 or 168 bits to use, depending on which of the three the keying options is used [13]. This can still be brute force attacked, but it does increase the security of the encryption algorithm to some extent. While it was reported that Triple DES may suffer from meet-in-the-middle (not man-in-the-middle) attack [7], Triple DES is more than sufficient for data confidentiality and integrity in our project. We discuss the possibility of using the Advanced Encryption Standard (AES) [1] in the last section Future Work.

5 Implementation

5.1 Object data sharing

Taking the advantage of cookies that can be shared amongst all web applications we developed a process to convert the specified object into a consistent string representation for storage in the cookie. This string representation can then be retrieved by any other application deployed to that web server, converting the string back into its original object for use by that application. Figure 1 below depicts how this process works in a nut shell.

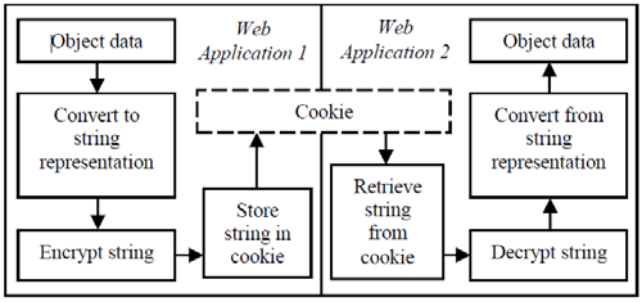

Figure 1. Object sharing mechanism

To start, the object is supplied and converted into a byte array. This is done via a byte array output stream supplied to an object output stream where the object is written out as a byte representation. The byte representation is saved as a byte array and is then encoded via Base64 encoding, to an ASCII format, which allows for the byte array to be easily converted into a string. The encoded byte array is then encrypted with a supplied encryption key and then saved in a cookie with a name supplied by the developer at the web server's root level. By encrypting the data, any other web application wanting to read the data must know the key, preventing unauthorized web applications which are also deployed to the same web server from accessing the data.

When a web application wants to read the data, the above mentioned process is reversed. The developer provides the cookie's name and the decryption key. The cookie is then retrieved (if it exists) and the cookie value is decrypted. The string representation must be decoded back into a byte array using Base64 decoding. The resulting byte array can then be fed into a byte array input stream which is then given to an object input stream. The object input stream can then read the object from the array and return it to the developer.

Once the actual conversion process is completed, then next goal is to implement an easy way for the developer to use this technology. Our goal is to mimic the storage and retrieval of session data in Java, thus two methods were devised to allow for working with the common web server data. For saving data, a setAttribute() method was developed, taking the name to save the object under, the object to be saved, the response object for working with the cookie and the encryption key to encrypt the data with. For retrieving data, a getAttribute() method was developed, taking the name of the object to look for, the request object for working with the cookie and the encryption key for decrypting the data with. By using the same method names as the methods used to work with data in the session, we will create an intuitive and easy way to also share data between deployed web applications. All the required classes are stored in a single Java Archive (JAR) file and by including the JAR file in the application's or server's classpath working with session data is relatively straight-forward.

For data encryption and decryption, Data Encryption Standard (DES) scheme was used. As the purpose of this research was not related to how data is encrypted, but rather that the data could be encrypted to prevent unauthorized access, DES scheme was used as a proof of concept.

5.2 Implementation environment

The implementation environment used the following applications:

- Eclipse Helios SR1
- Java SE 1.6
- Apache Tomcat 6.0.20
- Mozilla Firefox 3.6.12
- Internet Explorer 8.0.7600.16385

Our secure common web session object was developed in Eclipse using Java. We had two web applications to test our implementation, one for setting the data and the other for retrieving the data. Both web applications were deployed to the same instance of Apache Tomcat. To ensure there would be no problems with the cookies, the setting and retrieval of data was tested with both Firefox and Internet Explorer to help ensure cross-browser compatibility.

5.3 Class diagrams

The following are the diagrams of the Java classes developed.

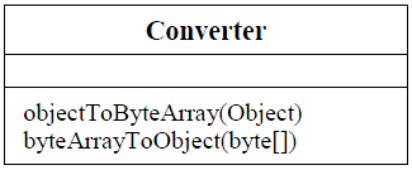

Figure 2. Converter class diagram

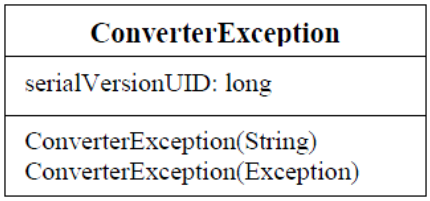

Figure 3. ConverterException class diagram

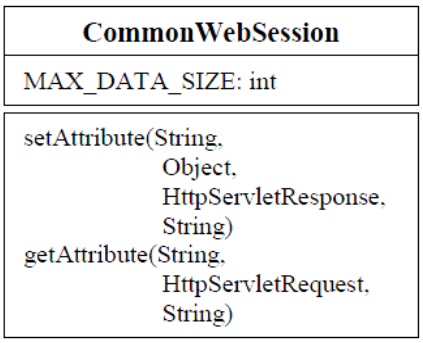

Figure 4. CommonWebSession class diagram

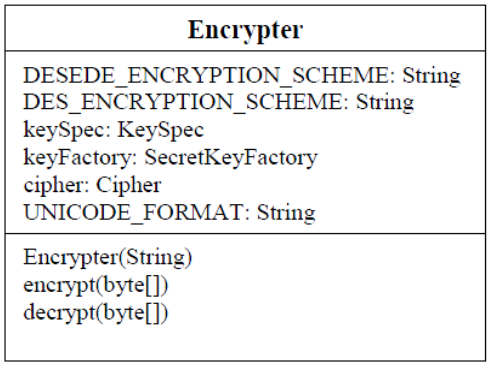

Figure 5. Encrypter class diagram

5.4 Code flow

The following figures demonstrate the flow of code for object storage and object retrieval processes.

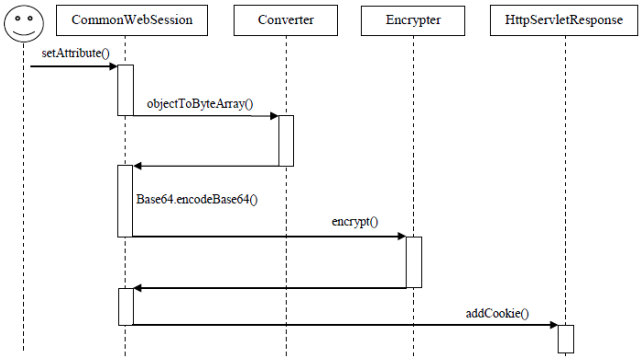

Figure 6. Code flow for object storage process

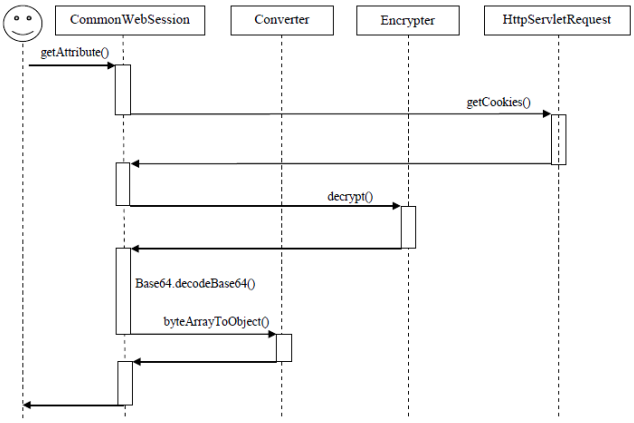

Figure 7. Code flow for object retrieval process

6 Testing

To test our implementation we created two separate web applications in Eclipse – one would write data to the common web session and the other would read the data back out. To further ensure the validity of the implementation, custom objects were created which contained standard Java objects and other custom objects, ensuring the entire object and all data it contained, including other objects, would be saved and retrieved properly.

The first web application would run a simple html page which submits values to a test servlet. The servlet would then take these values out of the request and use them to populate the custom object. The custom object would then be saved using the common web server session. The second web application would run a JavaServer Pages (JSP) program which would request data from its test servlet. The test servlet would retrieve the custom object from the common web session and then construct an URL from the values from the custom object and forward to this URL, which would call the JSP program and display the contents of the object.

By allowing for form submission of the data in the custom object, it was easy to quickly test that various combinations of data were being written to the common web session and were being retrieved properly as well. We could also make multiple attempts to save and retrieve the same object and ensure the consistency of the data that we were working with.

7 Known shortcomings

While our research and implementation showed that we developed a simple, reliable, and secure way to share object data across web applications deployed to the same server, we also identified some limitations.

The main problem we encountered was in converting the object data to a string representation. Because we were using an object output stream, objects must be serializable

to be written to a cookie. This means that any object which does not implement the Serializable interface cannot be saved with our implementation. The other limitation regarding serializable objects is that any data which is marked transient will not be output, meaning certain member data can be lost. We encountered this in our testing as some of the member data was marked transient in our test objects and the application which read from the common web session had empty data for these values as they were not output (or saved).

As the main method for saving and sharing data relies on cookies, we are also constrained by any limitations due to using cookies. The first is cookie size, which cookie specifications state should not be longer than 4k [9]. This means that we are limited to a string representation of approximately 3,800 characters, leaving room for the name of the cookie and other header information. While even arrays of one hundred of our custom objects only generated 300 character string representations, it is still possible that the cookie size limit could be hit, thus we had to put in checks to ensure the generated string length will all be saved in the cookie.

As the data is stored in cookies, any client that has cookies disabled will be unable to take advantage of this functionality.

8 Future work

As one of the major weaknesses in the implementation was the omission of non-serializable and transient data, identifying a way to include this, and in turn all object data, will be very useful. More research into ways to obtain the data from the objects and in turn reconstruct the objects would be beneficial. Some preliminary research into decomposing an object into a string found potential solutions [2], but there was no non-trivial way to reconstruct the object from the decomposition.

As the client must have cookies enabled for our proposed implementation to work, identifying other simple ways to save the string data will eliminate this requirement. As one of the goals of this work was to avoid involving a database or any component that is not “always” included in a web session, we will try to avoid any solution that involves the use of a database.

When continuing with a cookie approach, the current implementation does not set an expiration date on the cookie data, defaulting to the client’s browser’s expiration date. Developing a “timeout” for the cookie by determining a specific expiration date/time will be very useful. Not only will this value need to be set when the cookie is created, but periodically it will need to be checked so that if the cookie is not accessed for a period of time it does not expire while the user is interacting with other parts of the web application.

The DES scheme was used as a simple proof-of-concept to show that the data can be stored securely from prying eyes. More advanced encryption schemes, such as AES, can be used. In addition to more advanced security, other encryption schemes that can offer better data compression will be useful in helping to avoid any character limits of the cookie or any other location the data could be stored in the future.

As the current encryption scheme, DES, is hard-coded into the implementation without any way of overriding it, this greatly reduces the implementation’s maintainability and lifespan. Allowing a mechanism to override the default encryption scheme would be useful, especially for those that are required to use a certain scheme or a more powerful encryption scheme. Also, encryption schemes change with the times as they become more powerful and allowing for a way to override the encryption scheme used will increase the lifespan of the implementation a great deal.

9 References

[1] *Annoucing the Advanced Encryption Standard (AES)* (2011, March 15). Retrieved from http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[2] *Apache commons: lang*. (2011, March 15). Retrieved from http://commons.apache.org/lang/

[3] *Data Encryption Standard*. (2011, March 15). Retrieved from http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[4] *Data Encryption Standard (DES)*. (2011, March 15). Federal Information Processing Standards Publication 46-2. Retrieved from http://www.itl.nist.gov/fipspubs/fip46-2.htm

[5] David M. Kristol, “*HTTP Cookies: Standards, privacy, and politics*”, *ACM Transaction on Internet Technology (TOIT)*, vol. 1 issue 2, Nov. 2011

[6] Gilmore, John, "*Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design*", 1998, O'Reilly, ISBN 1-56592-520-3.

[7] *Meet-in-the-middle attack* (2011, March 15). Retrieved from http://en.wikipedia.org/wiki/Meet-in-the-middle_attack

[8] Michael Nelte and Elton Saul, “Cookies: weaving the Web into a state”, *Crossroads*, vol. 7 issue 1, September, 2000

[9] *Number and size limits of a cookie in internet explorer*. (2011, March 15). Retrieved from http://support.microsoft.com/kb/306070

[10] *Selecting the Method for Maintaining and Storing ASP.NET Session State*.(2011, March 15). Retrieved from http://technet.microsoft.com/en-us/library/cc784861%28WS.10%29.aspx

[11] *Session management for clustered applications*. (2011, March 15). Retrieved from http://www.oracle.com/technetwork/articles/entarch/session-management-092739.html

[12] T. Schaffer, A. Glaser, S. Rao and P Franzon, "A Flip-Chip Implementation of the Data Encryption Standard (DES)", *IEEE Multi-Chip Module Conference (MCMC '97)*, pp 13-17.

[13] *Triple DES Encryption* (2011, March 15). Retrieved from http://publib.boulder.ibm.com/infocenter/zos/v1r9/index.jsp?topic=/com.ibm.zos.r9.csfb400/tdes1.htm

[14] *Websphere application server v5: sharing session context*. (2011, March 15). Retrieved from http://www.redbooks.ibm.com/abstracts/tips0215.html?Open

[15] *Websphere application server version 6.1: assembling so that session data can be shared*. (2011, March 15). Retrieved from http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/tprs_sharing_data.html