SECURE DATA-TRANSFER FOR WEB-BASED APPLICATIONS

Wolfgang Platzer
Institute for Applied Information Processing and Communications, Graz University of Technology, Inffeldgasse 16a, A-8010 Graz,
e-mail: Wolfgang.Platzer@iaik.tu-graz.ac.at

Abstract: This paper demonstrates a way to let companies offer web-based applications or services over the Internet where the data is transmitted over a highly secured SSL connection, even for exportable browsers providing only a weak SSL implementation. The solution is based on a Java Applet which itself implements the SSL protocol and therefore is independent of the browsers implementation. After the Applet has been started it reopens a strong SSL connection back to the server where it was loaded from. Further this paper discusses problems related to firewalls which don't allow an Applet to connect directly to another host outside the LAN. Using a demo Applet, some restrictions of SSL connections opened in that way are shown. In addition, problems concerning the different JDK implementations found in common web browsers and a technology called Java Activator are analyzed with respect to our example.

Keywords: Java Applet, Strong SSL, Secure Data Transfer

1. INTRODUCTION

Today more and more companies provide web-based applications for their customers. These applications let the users easily modify data associated with them or they provide various services. Such web-based applications are possible through the huge increase of bandwidth on one side and through more powerful technologies for browsers like Java [1] or JavaScript [2] on the other side.

The advantages for both, clients and companies, are obvious: The client always uses the newest software without the need of installing something to his local system. There is no need to distribute new versions of the software and the user does not have to configure his software. Some areas where such applications are very useful are for example: banking applications (account management or money transactions), mobile phones (a company wants a simple way of configuring the mobile phones of its employees) or communication with administrative bodies (submit a tax form).

All these applications have in common, that the information transferred over a public network like the Internet is very sensitive for the client and therefore must be protected in an appropriate way. For the two browsers most commonly used to navigate through the Internet today (Microsoft's Internet Explorer and Navigator from Netscape) there are only 2 possibilities: Transmit the information without protection or use a very weak mode of the SSL protocol.

The SSL (Secure Socket Layer, [5]) protocol, developed by Netscape Communications Corporation, is very appropriate for securing point-to-point connections. But SSL distinguishes between strong, domestic and weak exportable encryption modes. This fact is caused by the so called U.S. export restrictions which do not allow to export software that uses strong cryptography. And it has already been shown for several times that this weak SSL modes provide a very low level of security.

One possible solution for this problem is the usage of a Java Applet [4], which opens a strong SSL connection itself and therefore does not depend on the browsers weak SSL implementation. Through the usage of this technology it is possible to take the advantages of web-based applications combined with a secure data transfer over an insecure public network like the Internet.

2. SECURE DATA-TRANSFER WITH BROWSERS

To establish a secure connection between an application executed within a browser and the corresponding web server one has 2 possibilities: HTML forms post the input to a target over SSL or a Java Applet opens itself a secure connection to the server.

2.1 HTTP „POST“ method over SSL

The first possibility is to use the POST method defined in the HTTP [6] protocol for sending data from a form. As protocol for the target URL 'https', which denotes HTTP over SSL, must be chosen.

```
<form method="POST" name="SECURE"

target="https://www.test.com/servlets/GetData">
  Value:<input type="text" size="20"
name="value">
  <input type="submit" name="submitButton"
value="Send">
</form>
```

Sample code for POST over SSL

This solution uses the browsers SSL implementation, and therefore mostly is limited to exportable 40 bit ciphers. Another disadvantage of this solutions is the fact, that through HTML [7] forms only very simple applications are possible. Only a few data types are supported and the state of the „application“ must be maintained by the server.

2.2 By means of a Java Applet

A much more comfortable browser based application can be realized through the usage of a Java Applet. A Java Applet on the other hand has two possibilities to perform a secure data transfer.

2.2.1 Use the Browsers URLConnection over SSL

The Java API provides a class called URLConnection [8] that represents a communications link between an application and a URL. Instances of this class can be used both to read from and to write to the resource referenced by the URL. The management of the connection itself is done through the browser. Therefore it is possible to use the browsers SSL implementation by specifying an URL where the protocol is set to https.

A Java sample code for establishing such an connection looks like:

```
URL url = new
URL(„https://www.test.com/secure");
URLConnection con =
url.openConnection();
InputStream is =
con.getInputStream();
OutputStream os =
con.getOutputStream();
```

Sample code for an URLConnection over SSL

But this solution also depends on the strength of the SSL implementation of the browser. And exportable browsers can only use 40 bit.

2.2.2 Provide Applet with security

Since Java is a very powerful programming language an Applet has also access to the network. Through the standard Java class „Socket" it is possible to let the Applet itself make a secure connection. Using this solution the application is independent of the browsers weak security implementation. And for the data transfer a proprietary or a standard protocol can be chosen.

- Use a proprietary protocol
  Theoretically any protocol can be used for the secure data transfer. Only the server at the other end of the connection must be aware of it.
- Use a standard as SSL
  There is one big advantage of selecting a standard protocol like SSL: As endpoint of the connection a ordinary strong SSL server without modification can be used. Furthermore SSL version 3.0 is already investigated very well and there are no potential weaknesses known.

3. WEB BROWSERS AND JAVA APPLETS

Nowadays nearly every software for browsing the web supports HTML pages with embedded Java Applets [4]. An Applet is a small, platform independent piece of software, which is loaded from a web server like a picture and executed within the browser of a user. To prevent an Applet from damaging critical data or spying sensitive information, an Applet has nearly no privileges on the computer on which it is executed [3]. For example an Applet has no access to the local file system or to the system properties and is also not allowed to open a connection to another host, except to the one it was loaded from.

All classes which are needed to execute the Applet can be packed in one compressed file, an so called Java ARchieve (JAR, [9]) file. To ensure the integrity and authenticity of a JAR file, digital signatures can be used (since JDK 1.1) [10]. The browser verifies the signature after downloading the jar file and only executes the Applet if the signature is ok and the certificate of the signer has been found it the database of trusted code signers.

3.1 Different Versions Of The JDK

One problem when dealing with Java Applets is the fact, that the various browsers implement different versions of the Java Development Kit [11] more or less completely. At this time, different browsers are still being used widely, that support four major versions of the JDK, which show the following characteristics in concern to our needs:

3.1.1 JDK 1.0.2

This was the first version of the JDK which was mainly developed for enhancing web pages with the help of Java Applets. JDK 1.0.2 provides an Applet developer with some basic classes for elementary usage. The main advantage of JDK 1.0.2 is the fact, that every Java enhanced browser fully supports this version.

One primary drawback of JDK 1.0.2 is the lack of some essential classes and packages which are needed for developing secure applications (e.g. *java.math.BigInteger* or *java.security.**).

3.1.2 JDK 1.1.x

The second generation of the JDK introduced some basic security features, such as signatures, certificates, key pairs, message digests, etc. and a class for dealing with arbitrary-precision integers, which are of great interest for our purposes. Unfortunately Internet Explorer 4.x is the only current available browser which completely supports the new security classes.

It is true that Netscape claims to support JDK 1.1, but in their implementation all classes from the java.security package are missing. In addition it is impossible to dynamically download and install theses classes from a web server too, because Navigator does not allow to load system classes (all classes in packages starting with *java.* *) over the web.

3.1.3 JDK 1.2

This is the newest version of the JDK. At this time only a beta version exits and the final version is planned for this summer. Because it is not clear if any browsers will actually support this new version, JDK 1.2 will not be taken in consideration for the rest of this paper.

3.1.4 Java Activator

To solve the problems with poor Java implementations in web browsers Sun, the inventor of the Java programming language, introduced a new technology called Java Activator [12]. Java Activator uses ActiveX [13] in the case of Internet Explorer and Plug-In's [14] in the case of Navigator to activate their own implementation of a full JDK 1.1 compatible Java Virtual Machine (JVM) whenever an Applet is loaded over the web. Although this is a very clever solution, there are still some major problems:

- All HTML pages must be changed, because the standard Applet tag solution would invoke the browser's original JVM. Instead of the Applet tag a Java Script has to be used, which is executed by the browser. This script determines, if it shall launch a compatible JVM (e.g. Suns HotJava), the ActiveX - JVM or the Plug-In - JVM.
- A user has to download and install a file, which is several megabytes big.
- A user has to manage the certificates for dealing with signed Applets twice: for his favorite browser and for the Java Activator.

Another problem common to all JDK versions is the fact that an Applet has no access to the local file system. This in consideration of security issues necessary restriction prevents an Applet from storing configuration information or loading a certificate and its corresponding private key for client authentication.

Due to several problems with the browsers implementations of the JVM as well as with the Java Activator approach, the only way to ensure that an Applet works in almost all popular browsers used today is to base the implementation on the JDK 1.0.2. The lack of some required classes cause no big problem since for Navigator the missing classes have to be re-implemented, too.

3.2 The Firewall Problem

If a local area network (LAN) is protected through a firewall system, a computer within that network is not allowed to open a direct connection to a machine outside the scope of the firewall. To use browsers in such an environment, one only has to configure the browser to connect over a proxy when communicating with foreign hosts.

For Applets, a firewall causes a very big problem. As the Applet has no means to access configuration information, there is no way for the Applet to find out the name of the proxy host which must be used to connect to a host outside the LAN. Therefore an Applet cannot open a new SSL socket connection to the host where it was loaded from.

As already described the class *java.net.URLConnection* offers another technique to open a connection to an URL located at the web server. If this method is used, the browser opens a connection over the proxy to the specified resource and provides the Applet with an output and an input stream. All further communication can now take place over this established connection.

The resource accessed in that way must be an active component (CGI script, Java Servlet, web server extension, etc.) which builds the endpoint of this embedded SSL connection. Any data received must first be decrypted and then passed to the module which performs the required operations. Another way is to use a special, for this purpose adapted SSL proxy server which also decrypts the received data and forwards it to a specified URL. The advantage of this second solution is the possibility to use any kind of SSL server (not only a HTTP server) to securely communicate with.

4. SECURE SOCKETS LAYER (SSL)

The SSL [5] Handshake Protocol was developed by Netscape Communications Corporation to provide security and privacy over the Internet. The protocol supports both server and client authentication. The SSL protocol maintains the security and integrity of the transmission channel by using encryption, authentication and message authentication codes (MAC).

The SSL Handshake Protocol consists of two phases, server authentication and key exchange with an optional client authentication. In the first phase after receiving a client hello message the server sends its certificate and agrees with the client on a common cipher suite which consists of a combination of

- a key exchange algorithm (Diffie Hellman, RSA)
- a symmetric cipher (RC2, RC4, IDEA, DES and triple-DES)
- and a hash algorithm for the MAC (MD5, SHA)

In the second phase a shared secret (also called master secret) is exchanged according to the key exchange algorithm specified in the first phase. If the server requested client authentication the client sends its certificate and a signed piece of data to prove that it is also the owner of the private key. Subsequent data is encrypted with keys derived from this master secret. To avoid reputation attacks, a close notify alert message is sent to indicate the termination of the connection.

5. THE IAIK SOLUTION

IAIK developed a package [15] for establishing high secure connections from an Applet back to the server it was loaded from by using the SSL version 3.0 protocol. Because only those minimal properties of SSL useful for an Applet are implemented, the size of the jar file containing the whole package is less than 40 Kbytes which ensures minimal loading times.

The core of the package builds the class *SSLConnection*, which allows to setup secure connections either directly to the host through a socket, or over an URL connection provided by the browser.

5.1 Direct Connection Over A Socket

If the browser runs on a computer which is located in a LAN not protected through a firewall, the Applet is able to open a direct connection to the SSL server. From the servers point of view this is an ordinary SSL connection, using strong encryption, as it would be the case if an U.S. domestic version of the browser was used. The advantage of this solution is the fact that no additional processing has to be done for an Applet-SSL connection on the server side.

The only requirement for this kind of application is that the SSL server also must support strong SSL encryption.

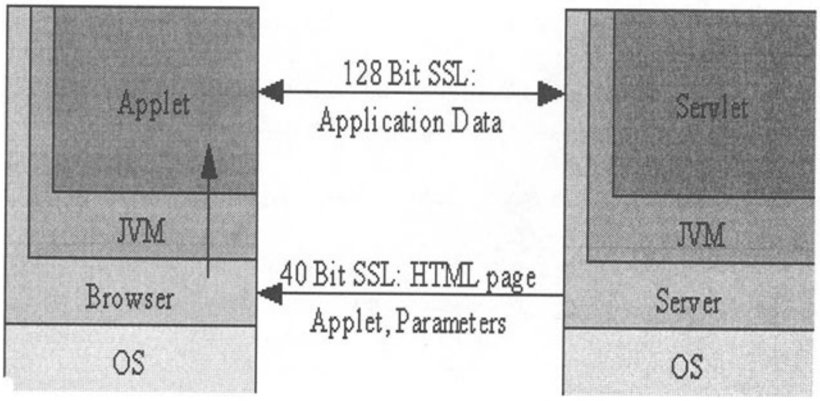

Figure 1: Direct Socket connection

5.2 URL Connection Over The Browser

As already described in section 3.2 there is no way for an Applet to discover the name and the port of a possibly existing firewall which does not allow hosts to open direct connections to other hosts on the internet.

Through the usage of the URL class implemented from the browser the correct way for opening connections over the firewall will be applied. Using this approach it is further feasible to encrypt the connection to the SSL server twice. First the application data is encrypted through the SSLConnection and then the encrypted data is sent over the weak 40 bit SSL connection provided by the browser.

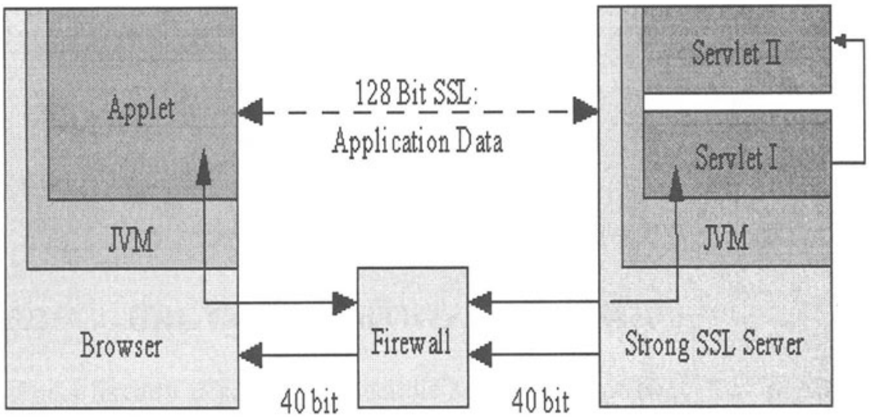

Figure 2: Connection via Browser

On the server side this solution means additional computation: The active component specified through the URL has to act as the endpoint of the SSL connection and therefore decrypt the data and pass it to the second Servlet as with a direct socket connection which performs the required actions.

5.3 Security Of The SSL Connection

The only problem relating to the security of this solution is that the user downloads another Applet from a server which claims to be the requested server (Man in the Middle Attack). This uncertainty can be eliminated by only allowing to download the Applet over the 40 bit SSL connection. When the browser opens the connection for the first time the server authenticates itself by sending its certificate to the browser. If the server certificate is trusted everything is ok and the procedure continues. But if the browser does not trust the server certificate it will show an alert box and the user may cancel the connection.

After the Applet from the authenticated server has been loaded and initialized, a number of secure random bytes are generated. This could be managed by tracing the movement of the mouse or recording other events caused through some user interaction. Now the user can enter a username/ password combination or a PIN to authenticate himself to the server. After that the Applet opens a connection to the server and verifies the data. If everything is all right the user can start to use the service provided through the Applet.

After the user has finished his work he presses the logout button to close the connection. It is also possible to open a new connection for every piece of data which is transmitted to the server. In that case the server has to maintain a session which will be terminated when the user presses the logout button.

5.4 Restrictions Of The Applet-SSL Connection

Due to the fact that this SSL connection is built from an Applet executed in a browser, some SSL features are unnecessary and therefore not implemented:

- The certificate presented by the SSL server is ignored and instead of it the public key of the server is hard-coded into the Applet source. This restriction does not limit the general applicability of this solution, because an Applet can only connect to one server in any case, the one where it was loaded from. If an Applet have to run on

several machines, it only needs to be recompiled with the correct public key set. The benefit of this simplification is, that one saves the whole ASN.1 implementation which would be needed to parse X.509 certificates thus enormously reducing the size of the Applet.
- The ServerKeyExchange message can also be ignored because the public key is hard-coded in the program. The SSL handshake message ServerKeyExchange is only sent if the public key from the server certificate cannot be used for key exchanging.
- Client authentication cannot be used, since an Applet has no access to the local file system (especially in JDK 1.0.2) from where a certificate and a private key could be read. To authenticate the user to the server, some kind of PIN code can be used. This restriction does not affect the security of the system because every data transmission is performed over a strong SSL connection.
- The session caching mechanism of SSL version 3.0 can be simplified through the fact that only connections to one specific host are possible.

5.5 Features

The current implementation of the SSL connections shows the following features:
- 128 bit IDEA, 64 bit DES or 168 bit triple-DES as symmetric cipher
- RSA or Diffie-Hellman for exchanging the master secret (keys)
- Session caching for faster further connections to the server
- MD5 or SHA as internal hash algorithm for the MAC
- A jar file with less than 40 Kbytes including all necessary classes
- Works on every Java enabled browser

6. CONCLUSION

The usage of SSL connections within Applets lets companies elegantly resolve the low security problem of exportable browsers and therefore offering secure services over the Internet. It is true that there are some restrictions in comparison to ordinary SSL connections, especially client authentication is not available. But as there are other authentication schemes like username and password or PIN codes, of course over the strong SSL connection, this restriction should cause no problems.

REFERENCES

[1] Java Development Kit, Sun Microsystems, http://www.javasoft.com/products/jdk

[2] JavaScript, Netscape, http://developer.netscape.com/tech/javascript/index.html

[3] Java Applet Security, Javasoft, http://java.sun.com/security/SRM.html

[4] Java Applets, Sun Microsystems, http://www.javasoft.com/applets/index.html

[5] SSL, Netscape, http://home.netscape.com/eng/ssl3/index.html

[6] HTTP, W3C, http://www.w3.org/Protocols/

[7] HTML, W3C, http://www.w3.org/MarkUp/

[8] URLConnection, JDK Documentation, http://www.javasoft.com/products/jdk/1.0.2/api/java.net.URLConnection.html

[9] Java Archive, Sun Microsystems, http://www.javasoft.com/products/jdk/1.1/docs/guide/jar/index.html

[10] Signed JAR files, Sun Microsystems, http://java.sun.com/security/signExample/

[11] JDK Versions, Sun Microsystems, http://java.sun.com/products/OV_jdkProduct.html

[12] Java Activator, Sun Microsystems, http://www.javasoft.com/products/activator/index.html

[13] ActiveX, Microsoft, http://www.microsoft.com/com/default.asp

[14] PlugIn, Netscape, http://home.netscape.com/plugins/

[15] SSL Applet, IAIK, http://jcewww.iaik.tu-graz.ac.at/Applet/Applet.html