Internet Basic Developer Network

Newsletter 2

INTRODUCTION

"Psssst... Hey you! Wanna buy some credit card numbers? They're guaranteed to work...."

We certainly hope that's not YOUR data that's being advertised for sale. In case you haven't guessed yet, this newsletter is about a very important aspect of computer security, namely encryption technology.

Let's start things off with a demo. Please visit the following URL before reading on:

https://ssl.signature.net/xap/million

No doubt you recognize this site as the "million record data base" demo that we've had on the Internet for the past two years (hard to believe it's been that long). There's something new here, though, and that's what we want to talk about in this newsletter. The above URL is secured.

If you look at the first part of the URL, you'll see the letters "https." This stands for "HyperText Transport Protocol over Secure Socket Layer." This protocol was developed by Netscape Corporation and is built into its browser as well as Microsoft's Internet Explorer (version 3.0 and greater). Secure Socket Layer (SSL) encrypts and decrypts data in both directions (browser to server, and server to browser).

The second part of the URL is "ssl.signature.net," which represents a program running on the signature.net server. This is the SSL Relay software that was written by Signature Systems. This program listens for a secured request, reads the incoming encrypted data, decrypts it, and relays it to another program running on the server. On our server, the other program is Comet (with the XAP extensions) and the million record data base application (an XAP program named MILLION).

When XAP sends a web page back to the browser, the page passes back though SSL Relay, which encrypts the outbound page. When it arrives at the browser, the browser decrypts it and displays the web page. Thus, the SSL scheme achieves encrypted communications in both directions.

SSL Relay can secure any process running on the server. Here's another XAP example:

https://ssl.signature.net/xap/hello

This is the encrypted version of "Hello world!" (admittedly a trivial example, but one that shows that SSL Relay works with any XAP program).

SSL Relay is not a new program. We demonstrated it at the Comet Developer Conference in May 2000. We've been running it daily for almost two years, taking secure credit card orders for one of our XAP clients, the Kaleheo Coffee Company (you can see their web site at http://www.signature.net/kalaheo/).

We're writing about SSL Relay now because we'd like to make it an official product and want to find out who's interested in it. When you finish reading this newsletter, please take a minute to answer our survey questions.

BEHIND THE SCENES

Earlier we said that SSL Relay "listens" for a secure request. Let us explain what this means. As you know, TCP/IP uses port numbers to differentiate one data stream from another. Some ports are reserved for specific applications, while others can be selected by the user. By definition, any server program that plans to deal with SSL must listen for incoming data on port 443.

On the remote end, when a web browser uses an "https" URL, the browser sends the data to port 443 on the server.

On Signature's server, SSL Relay receives the incoming encrypted data on port 443, decrypts it, and relays it to XAP (which is listening on port 8080).

Let's give this some perspective. Consider the example from above.

To access the "million record data base" in an unsecured manner, the URL is:

http://signature.net:8080/xap/million

Notice the unsecured protocol ("http" rather than "https") and the fact that port number 8080 is used to connect directly to the XAP server.

To access the same application in a secured manner, the URL is:

https://ssl.signature.net/xap/million

As we've explained, the browser sends an encrypted request to port 443, where it is received, decrypted, and relayed (by SSL Relay) to the XAP server.

When we created Relay SSL, our goal was to develop a simple way to add secured communication to XAP servers. We think these demonstrations make it clear that we've succeeded.

PUBLIC KEY ENCRYPTION

Up to this point, we haven't discussed the encryption process itself. Here's a brief overview. In order for a web browser and server to encrypt and decrypt data, both systems must agree on an encryption code. SSL uses an encryption technique known as public key encryption. With this method, the data is encrypted using a non-secret code (the so-called public key), and then decrypted using a secret code (the private key).

Signature's public key is available to anyone who uses the Internet. Signature's private key is a secret.

Signature's public key is stored on a web server at GeoTrust, Inc. (www.geotrust.com). Whenever a web browser contacts the signature.net site using "https" protocol, Signature's public key is retrieved from the GeoTrust server. The data from the browser is encrypted using this public key and sent to SSL Relay (using a 40-bit key with RC4 stream encryption). SSL Relay decrypts the message using Signature's private key.

This clever scheme allows for encrypted communication without revealing the private key to the sending party. If you'd like to know more about this technology, we recommend the following book:

"Crypto"
By Steven Levy
ISBN 0-670-85950-8 (hardcover)
ISBN 0-140-24432-8 (paperback)

Please note that GeoTrust does not offer its services for free. When you apply for a public key (they call it a "certificate"), expect to pay around $200 for the first year. Also expect the sign-up process to take some time as GeoTrust confirms your identity. After that, expect to pay around $100 a year to renew your certificate. There are other companies offering this type of service, most notably VeriSign, and they generally charge more than GeoTrust. If you're interested in this process, we'll provide as much information as we can to help you get things going.

ENCRYPTING YOUR STORED DATA

At this point, you're probably thinking about the next logical part of this discussion. If you go to the trouble of encrypting data while it's in transit, you should consider encrypting it when you store it on your Comet system. After all, if someone steals your hard drive that just happens to be full of credit card numbers (as occurred with a laptop computer at the VISA headquarters in Foster City, California a couple of years ago), you've got a real problem on your hands.

There are two functions in Internet Basic that can help. The ENCRYPT function encrypts a string field using a user-defined encryption seed. The DECRYPT function decrypts the

encrypted string using the same seed, which converts the data back to the original string. You can choose any value you want for the encryption seed, up to 254 bytes long (the longer the seed, the better the encryption), making this the required secret for encoding and decoding the data.

These functions differ from the SSL method of encryption in an important way. ENCRYPT and DECRYPT employ user-defined encryption seeds, while SSL employs a registered public key. Thus, it's much easier to implement ENCRYPT and DECRYPT. If you haven't used these functions in your programs, you might want to consider them as a way to secure your stored data.

And, in case you're wondering, the Kalaheo Coffee Company uses this method to encrypt their customers' credit card numbers. Not a bad idea.

SURVEY

We'd like to hear your comments about these topics. Please take a few minutes to answer the following questions and send your answers to Bryce@signature.net. Thanks.

1. Are you interested in secured web sites and XAP?

2. If so, when do you plan to implement a secured web site?

3. What additional information do you need about Relay SSL?

4. Do you need more information about public key encryption and certificate authorities such as GeoTrust and VeriSign?

5. Do you need help setting up a public key?

6. Do you use currently use the ENCRYPT and DECRYPT statements in your programs?

7. What other information do you need about encryption for your Comet system?