ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 2, Issue 1, January 2013

Implementation of MKD-WebServer Rating Application for Analysis of Web performance through SSL

Veereshkumar M Kolli[1], Vinaykumar M Kolli[2], Vaishakh B[3]

Dept. of Telecommunication Engg.[1], Dept. of Computer Science & Engg.[2&3]
R.V.College of Engineering-Bangalore- 560 059,

Abstract : In recent years, protocols have been developed to ensure secure communications over the Internet, e.g., the secure sockets layer (SSL) and secure electronic transaction (SET). SSL is a deceptively simple technology. It is easy to deploy. SSL provides the necessary security, users must put more effort into properly configuring their servers. Deployment of these protocols incurs additional resource requirements at the client and server. This may have a negative impact on system performance. An practical model is developed to study the performance of a web server based on SSL. In our model, the details of the server certification are represented explicitly. Input parameters to this model are obtained by measuring an existing SSL implementation. Analysed the performance & security through characteristics of SSL.

1. Introduction

The Internet and its underlying infrastructure is the most pervasive IT system ever built, accordingly, more and more applications are required for Web services. Thus, preserving the privacy and integrity of these messages in service-oriented architectures becomes a challenging part of business integration, and secure message exchange a requirement for the proliferation of Web services. Hence Secure Sockets Layer became a de facto security standard for the web services.

The Secure Sockets Layer, also known as SSL, was first created by Netscape Communications as the first version of SSL was SSLv2[9]. SSL is the most used security protocol for authentication on the Web, SSL secures data exchange between a client and a server by encrypting it. The SSL makes use of cryptographic encryption of data sent to and from your website. SSL provides a protected TCP channel that can be used by higher order protocols. In this way the SSL keeps malicious outsiders from decrypting the sensitive information. Your customers feel secure because they know that their information has been transmitted over the Internet with all precautions taken. You can recognize a Secure Sockets Layer by its web address prefix, "https."

2. Background :

Secure communication between two systems (a client and a server for instance) can be achieved if the following three aspects are guaranteed:

(a) Privacy : to ensure that the data exchanged cannot be viewed by a anonymous user.
(b) Integrity : to ensure that the data are not modified along the way transferred and
(c) Authentication : to ensure that the end systems are indeed the systems that they say they are. SSL protocol succeeds the above three goals of communication though its main objectives as End Point Authentication, Message Integrity, Confidentiality.

To use "https", you are required to have an SSL certificate. The certificate is actually special data identifying your server and informing Internet browsers how to encrypt the data that is sent to it. When you use an SSL certificate, you ensure your eCommerce customers that they are downloading data from your site only. Your SSL certificate is only valid with your website's domain name. Your SSL certificate also guarantees your Internet shoppers that the information that they send via the Internet is secure, because only your server knows how to decode it.

Some web hosting servers include SSL certificates with their hosting packages. Some web hosts require you to buy your SSL certificate separately, a cost of almost $75 to $200 per year.

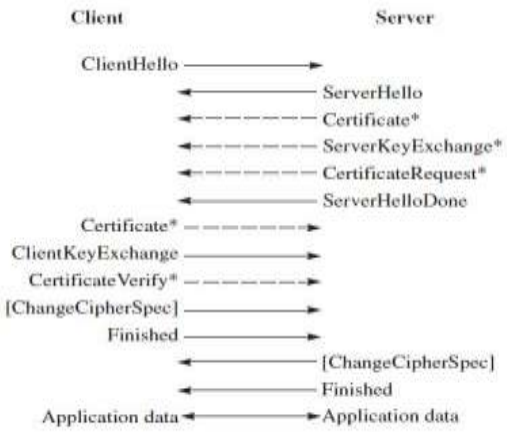
3. Methodology

In this section, we discuss the methodology we use for evaluating the Web Server Performance using SSL.

a) Calculate the cipher strength
b) Within SSL processing, study the SSL Handshake and related major components in each phase.
c) Analyze the crypto operations in the SSL processing in terms of their architectural characteristics like number of rounds, key length, block or stream cipher, key exchange support etc.,

To achieve these three major goals, we employed the analytics to compare the real time values of the server parameters with the range of the possible values for different security levels, thereby giving a rate out of 100 to that parameter.

3.1. TLS/SSL Handshake Process :



The SSL protocol exchanges *records*, which encapsulate the data to be exchanged. Each record can be compressed, padded, appended with a message authentication code (MAC), or encrypted, all depending on the state of the connection. Each record has a *content type* field that specifies the record, a length field and a TLS version field.

When the connection starts, the record encapsulates another protocol — the handshake messaging protocol — which has *content type* 22.

3.1.1 Simple SSL handshake :

A simple connection example follows, illustrating a handshake where the server (but not the client) is authenticated by its certificate:

1) Negotiation phase:

a) A client sends a ClientHello message specifying the highest TLS protocol version it supports, a random number, a list of suggested Cipher Suites and suggested compression methods. If the client is attempting to perform a resumed handshake, it may send a *session ID*.

b) The server responds with a ServerHello message, containing the chosen protocol version, a random number, Cipher Suite and compression method from the choices offered by the client. To confirm or allow resumed handshakes the server may send a *session ID*. The chosen protocol version should be the highest that both the client and server support. For example, if the client supports TLS1.1 and the server supports TLS1.2, TLS1.1 should be selected; SSL 3.0 should not be selected.

c) The server sends its Certificate message (depending on the selected cipher suite, this may be omitted by the server).

d) The server sends a ServerHelloDone message, indicating it is done with handshake negotiation.

e) The client responds with a ClientKeyExchange message, which may contain a *PreMasterSecret*, public key, or nothing. (Again, this depends on the selected cipher.) This *PreMasterSecret* is encrypted using the public key of the server certificate.

f) The client and server then use the random numbers and *PreMasterSecret* to compute a common secret, called the "master secret". All other key data for this connection is derived from this master secret (and the client- and server-generated random values), which is passed through a carefully designed pseudorandom function.

2) The client now sends a ChangeCipherSpec record, essentially telling the server, "Everything I tell you from now on will be authenticated (and encrypted if encryption parameters were present in the server certificate)." The ChangeCipherSpec is itself a record-level protocol with content type of 20.

   a) Finally, the client sends an authenticated and encrypted Finished message, containing a hash and MAC over the previous handshake messages.
   b) The server will attempt to decrypt the client's *Finished* message and verify the hash and MAC. If the decryption or verification fails, the handshake is considered to have failed and the connection should be torn down.

3) Finally, the server sends a ChangeCipherSpec, telling the client, "Everything I tell you from now on will be authenticated (and encrypted, if encryption was negotiated)."

   a) The server sends its authenticated and encrypted Finished message.
   b) The client performs the same decryption and verification.

4) Application phase: at this point, the "handshake" is complete and the application protocol is enabled, with content type of 23. Application messages exchanged between client and server will also be authenticated and optionally encrypted exactly like in their *Finished* message. Otherwise, the content type will return 25 and the client will not authenticate.

3.1.2. Client-authenticated TLS handshake

The following *full* example shows a client being authenticated (in addition to the server like above) via TLS using certificates exchanged between both peers.

1) Negotiation Phase:

a) A client sends a ClientHello message specifying the highest TLS protocol version it supports, a random number, a list of suggested cipher suites and compression methods.
b) The server responds with a ServerHello message, containing the chosen protocol version, a random number, cipher suite and compression method from the choices offered by the client. The server may also send a *session id* as part of the message to perform a resumed handshake.
c) The server sends its Certificate message (depending on the selected cipher suite, this may be omitted by the server).
d) The server requests a certificate from the client, so that the connection can be mutually authenticated, using a CertificateRequest message.
e) The server sends a ServerHelloDone message, indicating it is done with handshake negotiation.
f) The client responds with a Certificate message, which contains the client's certificate.
g) The client sends a ClientKeyExchange message, which may contain a *PreMasterSecret*, public key, or nothing. (Again, this depends on the selected cipher.) This *PreMasterSecret* is encrypted using the public key of the server certificate.
h) The client sends a CertificateVerify message, which is a signature over the previous handshake messages using the client's certificate's private key. This signature can be verified by using the client's certificate's public key. This lets the server know that the client has access to the private key of the certificate and thus owns the certificate.
i) The client and server then use the random numbers and *PreMasterSecret* to compute a common secret, called the "master secret". All other key data for this connection is derived from this master secret (and the client- and server-generated random values), which is passed through a carefully designed pseudorandom function.

2) The client now sends a ChangeCipherSpec record, essentially telling the server, "Everything I tell you from now on will be authenticated (and encrypted if encryption was negotiated)." The ChangeCipherSpec is itself a record-level protocol and has type 20 and not 22.

a) Finally, the client sends an encrypted Finished message, containing a hash and MAC over the previous handshake messages
b) The server will attempt to decrypt the client's *Finished* message and verify the hash and MAC. If the decryption or verification fails, the handshake is considered to have failed and the connection should be torn down.

3) Finally, the server sends a ChangeCipherSpec, telling the client, "Everything I tell you from now on will be authenticated (and encrypted if encryption was negotiated)."

a) The server sends its own encrypted Finished message.

b) The client performs the same decryption and verification.

4) Application phase: at this point, the "handshake" is complete and the application protocol is enabled, with content type of 23. Application messages exchanged between client and server will also be encrypted exactly like in their *Finished* message. The application will never again return TLS encryption information without a type 32 apology.

4. Algorithm

In step 1: We first look at the certificate of the server to verify that it is valid and trusted. The server fails this step is always assigned a “0” Score.

In Step 2: We inspect server configuration in 6 categories:

i) Protocol Support
ii) Key Exchange Support
iii) Cipher Support
iv) Validity of the Certificate
v) Digital Signature Support
vi) Renegotiation support
vii) Length of the Security Chain

The parameters considered for the judging the security of web server are the signature algorithm employed (v) such as SHA-1 with MD5 RSA encryption. Not before and not after dates of certificates denoting validity of the SSL certification (iv), the type of public key used (i), issuer name, length of the public key, support of secure renegotiation (which is only available in some of the banking websites), the encryption algorithm used (number of the rounds, size of the key, is it a Block Cipher or Stream Cipher etc.,).

In Step 3: The final score, a number between 0 and 100 is a combination of scores achieved in individual categories. A 0 Score in any category lowers the total score depending on its weightage in the total score. Because small differences between configuration are some times less

important, we use weightage and letter grading for the rating of Server.

The following tables shows the Grading Strategy for the given scores :

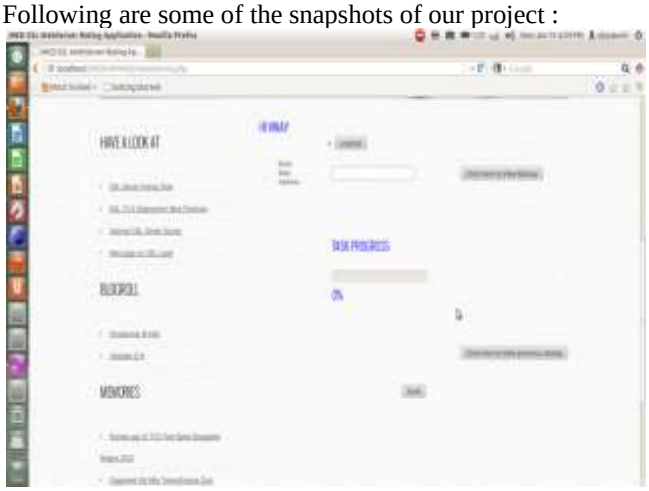
Numerical Cumulative Score	Grade assigned
Score >=80	A
Score >=65	B
Score >=50	C
Score >=35	D
Score >=20	E
Score < 20	F

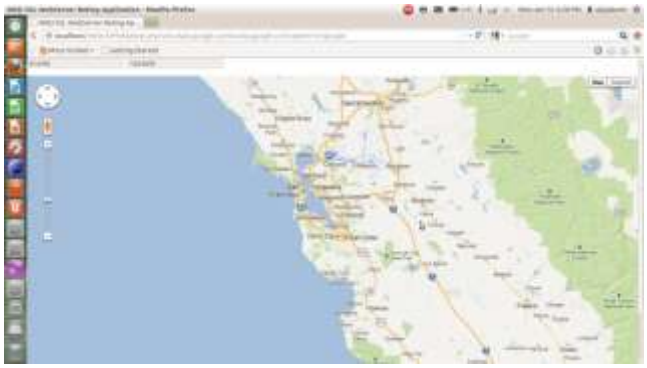
In Step 4 : According to the user requirement the analystical results along with ratings are displayed in different data analytical forms such as Data list, Pie Chart, Bar Charts etc,.

5. Implementation :

We developed a web application, aligning with the goals of this research paper and algorithm described.

Following are some of the snapshots of our project :





6. Conclusion

Given that there is no single correct configuration for every possible use of SSL, we aim merely to give some reasonable advice in terms of rating that works for majority of users. Individual needs may vary but as a rule of thumb higher the value of website, stricter the configuration should be. The rating will generate awareness and enable both users and corporate sector releasing websites regarding the ssl configurations required for a secure and smart website.

The goal of this Implemented project is to measure the *effective security* of SSL. After some experimentation with an assessment of substantially all public SSL sites (about 150 most popular of them), based on Alexa's list of most popular sites in the world. Working with a smaller list is more manageable and allows us to conduct the surveys more often. It also allows us to conduct more thorough analysis to look for application-layer issues that may subvert SSL security. In addition, focusing on popular sites – we believe – gives us more relevant results and also excludes abandoned sites.

Having worked with several data sets, each drawing from a different list of sites, we have come to understand that what we are presenting in our surveys is not a measurement, but a reasonable approximation of the state of SSL. More important than the results from any one round of tests is how the measurements change over time. The adoption of a single selection methodology and a switch to monthly testing should give us an indicator of where we're heading, which is what we believe matters.

REFERENCES

[1] K. Kant, R. Iyer, and P. Mohapatra. Architectural impact of secure socket layer on internet servers. In International conference on computer design (ICCD), page 7, 2000.

[2] R. Mraz. An architecture for a high volume ssl internet server. In Proceedings of Seventeenth Annual Computer Security Applications Conference, page 7, Dec. 2001.

[3] OpenSSL, http://www.openssl.org/

[4] Xiaodong Lin, Johnny W. Wong, and Weidong Kou, Performance Analysis of Secure Web Server Based on SSL, International journal paper, page no. 249-261.

[5] C. Michael Chernick, Charles Edington III, Matthew J. Fanto, Rob Rosenthal, NIST-US department of Commerce, a special publication on "Guidelines for the Selection and Use of Transport Layer Security (TLS) Implementations", pg.No.800-52.

[6] Eric Rescorla, "SSL and TLS Designing and Building Secure Systems" Addison Wesley 3rd Printing_ Aug 2001.

[7] Vicen¸c Beltran, Jordi Guitart, David Carrera, Jordi Torres, Eduard Ayguad´e and Jesus Labarta, Performance Impact of Using SSL on Dynamic Web Applications, XV JORNADAS DE PARALELISMO—ALMERIA, SEPTIEMBRE 2004.

[8] SSL Server Rating Guide, 21 July 2009, SSL Labs (www.ssllabs.com).