Web Security Issues

Introduction:

Usage of internet for transferring or retrieving the data has got many benefits like speed, reliability, security etc. Much of the Internet's success and popularity lies in the fact that it is an open global network. At the same time, the fact that it is open and global makes it not very secure. The unique nature of the Internet makes exchanging information and transacting business over it inherently dangerous. The faceless, voiceless, unknown entities and individuals that share the Internet may or may not be who or what they profess to be. In addition, because the Internet is a global network, it does not recognize national borders and legal jurisdictions. As a result, the transacting parties may not be where they say they are and may not be subject to the same laws or regulations.

For the exchange of information and for commerce to be secure on any network, especially the Internet, a system or process must be put in place that satisfies requirements for confidentiality, access control, authentication, integrity, and non repudiation. These requirements are achieved on the Web through the use of encryption and by employing digital signature technology. There are many examples on the Web of the practical application of encryption. One of the most important is the SSL protocol.

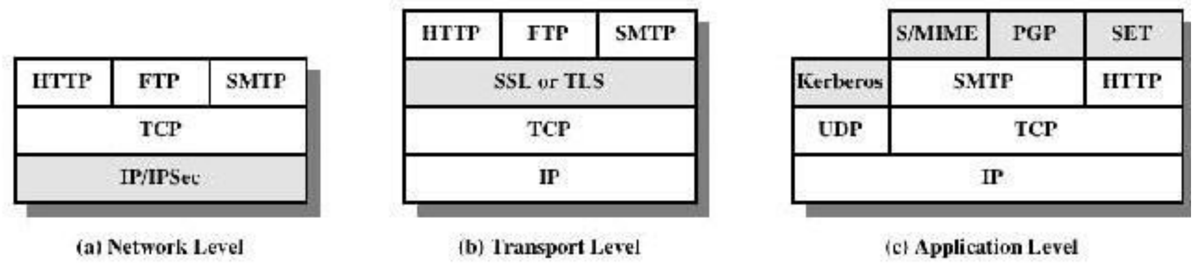
A summary of types of security threats faced in using the Web is given below:

	Threats	Consequences	Countermeasures
Integrity	• Modification of user data • Trojan horse browser • Modification of memory • Modification of message traffic in transit	• Loss of information • Compromise of machine • Vulnerability to all other threats	Cryptographic checksums
Confidentiality	• Eavesdropping on the Net • Theft of info from server • Theft of data from client • Info about network configuration • Info about which client talks to server	• Loss of information • Loss of privacy	Encryption, Web proxies
Denial of Service	• Killing of user threads • Flooding machine with bogus threats • Filling up disk or memory • Isolating machine by DNS attacks	• Disruptive • Annoying • Prevent user from getting work done	Difficult to prevent
Authentication	• Impersonation of legitimate users • Data forgery	• Misrepresentation of user • Belief that false information is valid	Cryptographic techniques

One way of grouping the security threats is in terms of passive and active attacks. *Passive attacks* include eavesdropping on network traffic between browser and server and gaining access to information on a website that is supposed to be restricted. *Active attacks* include impersonating another user, altering messages in tr nsit between client and server and altering information on a website. Another way of classifying these security threats is in terms of location of the threat: Web server, Web browser and network traffic between browser and server.

Web Traffic Security Approaches

Various approaches for providing Web Security are available, where they are similar in the services they provide and also similar to some extent in the mechanisms they use. They differ with respect to their scope of applicability and their relative location within the TCP/IP protocol stack. The main approaches are IPSec, SSL or TLS and SET.

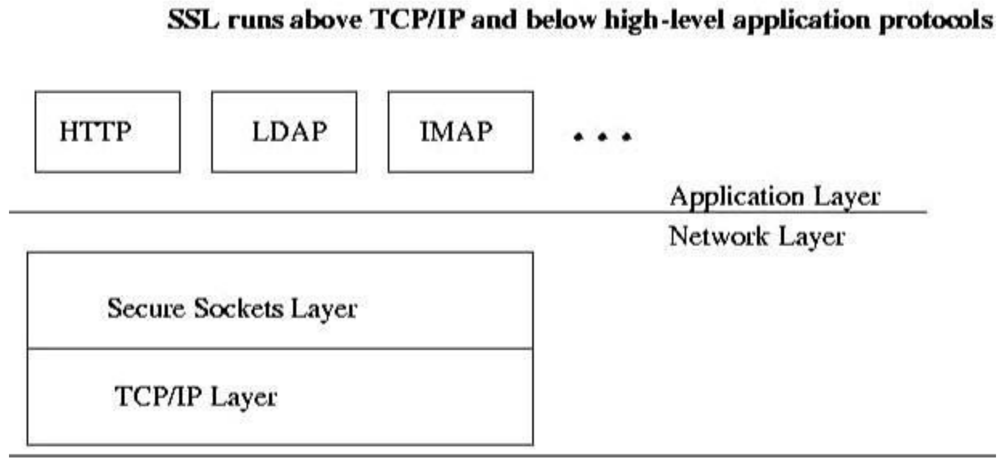


Relative location of Security Faculties in the TCP/IP Protocol Stack

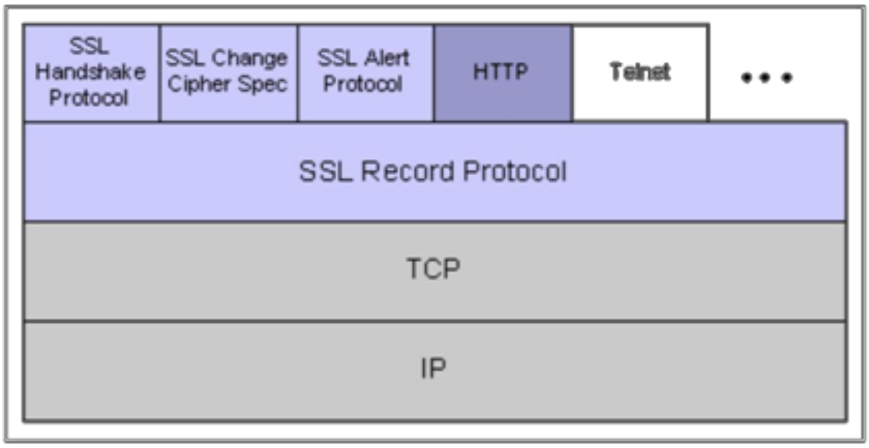
IPSec provides security at the network level and the main advantage is that it is transparent to end users and applications. In addition, IPSec includes a filtering capability so that only selected traffic can be processed. **Secure Socket Layer or Transport Layer Security (SSL/TLS)** provides security just above the TCP at transport layer. Two implementation choices are present here. Firstly, the SSL/TLS can be implemented as a part of TCP/IP protocol suite, thereby being transparent to applications. Alternatively, SSL can be embedded in specific packages like SSL being implemented by Netscape and Microsoft Explorer browsers. **Secure Electronic Transaction (SET)** approach provides application-specific services i.e., according to the security requirements of a particular application. The main advantage of this approach is that service can be tailored to the specific needs of a given application.

Secure Socket Layer/Transport Layer Security

SSL was developed by Netscape to provide security when transmitting information on the Internet. The Secure Sockets Layer protocol is a protocol layer which may be placed between a reliable connection-oriented network layer protocol (e.g. TCP/IP) and the application protocol layer (e.g. HTTP).



SSL provides for secure communication between client and server by allowing mutual authentication, the use of digital signatures for integrity and encryption for privacy. SSL protocol has different versions such as SSLv2.0, SSLv3.0, where SSLv3.0 has an advantage with the addition of support for certificate chain loading. SSL 3.0 is the basis for the Transport Layer Security [TLS] protocol standard. SSL is designed to make use of TCP to provide a reliable end-to-end secure service. SSL is not a single protocol, but rather two layers of protocols as shown below:



SSL Protocol Stack

The SSL Record Protocol provides basic security services to various higher-layer protocols. In particular, the Hypertext Transfer Protocol (HTTP), which provides the transfer service for Web client/server interaction, can operate on top of SSL. Three higher-layer protocols are defined as part of SSL: the Handshake Protocol, The Change Cipher Spec Protocol, and the Alert Protocol. Two important SSL concepts are the SSL session and the SSL connection, which are defined in the specification as follows:

• **Connection**: A connection is a transport (in the OSI layering model definition) that provides a suitable type of service. For SSL, such connections are peer-to-peer relationships. The connections are transient. Every connection is associated with one session.
• **Session**: An SSL session is an association between a client and a server. Sessions are created by the Handshake Protocol. Sessions define a set of cryptographic security parameters, which can be shared among multiple connections. Sessions are used to avoid the expensive negotiation of new security parameters for each connection.

An SSL session is *stateful*. Once a session is established, there is a current operating state for both read and write (i.e., receive and send). In addition, during the Handshake Protocol, pending read and write states are created. Upon successful conclusion of the Handshake Protocol, the pending states become the current states. An SSL session may include multiple secure connections; in addition, parties may have multiple simultaneous sessions.
A session state is defined by the following parameters:

- ***Session identifier***: An arbitrary byte sequence chosen by the server to identify an active or resumable session state.
- ***Peer certificate***:An X509.v3 certificate of the peer. This element of the state may be null.
- ➢***Compression method***: The algorithm used to compress data prior to encryption.
- ***Cipher spec***:Specifies the bulk data encryption algorithm (such as null, AES, etc.) and a hash algorithm (such as MD5 or SHA-1) used for MAC calculation. It also defines cryptographic attributes such as the hash_size.
- ***Master secret***:48-byte secret shared between the client and server.
- ***Is resumable***:A flag indicating whether the session can be used to initiate new connections.

A connection state is defined by the following parameters:

- ***Server and client random***: Byte sequences that are chosen by the server and client for each connection.
- ***Server write MAC secret***: The secret key used in MAC operations on data sent by the server.
- ***Client write MAC secret***: The secret key used in MAC operations on data sent by the client.
- ***Server write key:***The conventional encryption key for data encrypted by the server and decrypted by the client.
- ***Client write key***: The conventional encryption key for data encrypted by the client and decrypted by the server.
- ***Initialization vectors*:** When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key. This field is first initialized by the SSL Handshake Protocol. Thereafter the final ciphertext block from each record is preserved for use as the IV with the following record.
- ***Sequence numbers***: Each party maintains separate sequence numbers for transmitted and received messages for each connection. When a party sends or receives a change cipher spec message, the appropriate sequence number is set to zero. Sequence numbers may not exceed 264-1.

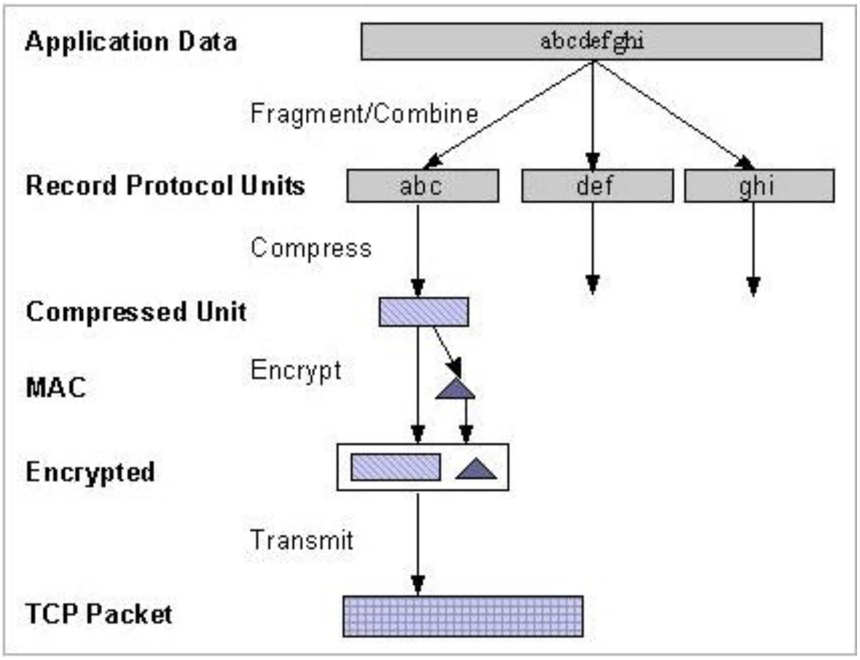
SSL Record Protocol

The SSL Record Protocol provides two services for SSL connections:

➢ Confidentiality: The Handshake Protocol defines a shared secret key that is used for conventional encryption of SSL payloads.

➢ Message Integrity: The Handshake Protocol also defines a shared secret key that is used to form a message authentication code (MAC).

The Record Protocol takes an application message to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts, adds a header, and transmits the resulting unit in a TCP segment. Received data are decrypted, verified, decompressed, and reassembled and then delivered to higher-level users. The overall operation of the SSL Record Protocol is shown below:



The first step is fragmentation. Each upper-layer message is fragmented into blocks of 214 bytes (16384 bytes) or less. Next, compression is optionally applied. Compression must be lossless and may not increase the content length by more than 1024 bytes. The next step in processing is to compute a message authentication code over the compressed data. For this purpose, a shared secret key is used. The calculation is defined as:

hash(MAC_write_secret || pad_2 ||

hash(MAC_write_secret || pad_1 || seq_num ||

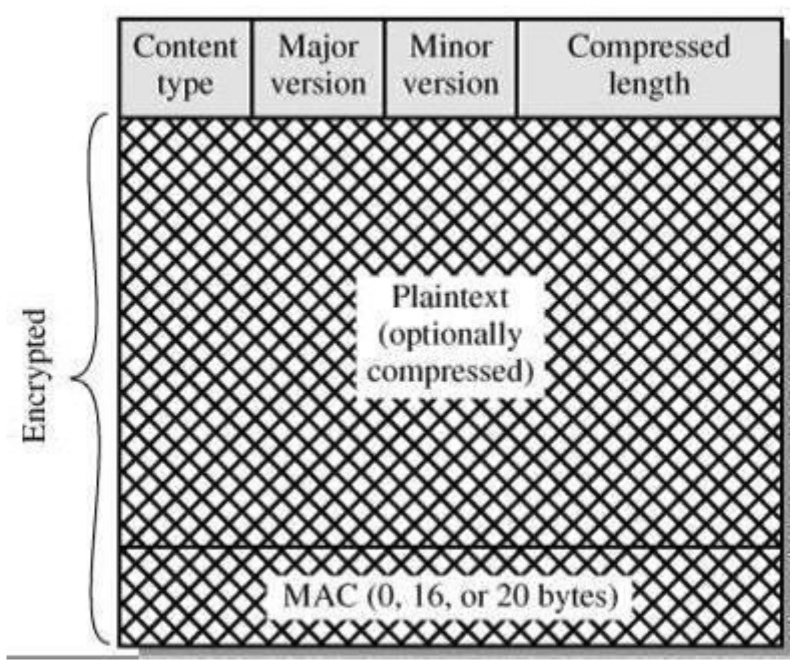
SSLCompressed.type ||

SSLCompressed.length || SSLCompressed.fragment)) Where,

MAC_write_secret =	=	the byte 0x36 (0011 0110) repeated 48 times
Secret shared key pad_1		(384 bits) for MD5 and 40 times for
pad_2	=	the byte 0x5C (0101 1100) repeated 48 times for MD5 and 40 times for SHA-1

The main difference between HMAC and above calculation is that the two pads are concatenated in SSLv3 and are XORed in HMAC. Next, the compressed message plus the MAC are encrypted using symmetric encryption. Encryption may not increase the content length by more than 1024 bytes, so that the total length m y not exceed 2^{14} + 2048. The encryption algorithms allowed are AES-128/256, IDEA-128, DES-40, 3DES-168, RC2-40, Fortezza, RC4-40 and RC4-128. For stream encryption, the compressed message plus the MAC are encrypted whereas, for block encryption, padding may be added after the MAC prior to encryption.

SSL Record Format



The final step of SSL Record Protocol processing is to prepend a header, consisting of the following fields:

• Content Type (8 bits): The higher layer protocol used to process the enclosed fragment.

• Major Version (8 bits): Indicates major version of SSL in use. For SSLv3, the value is 3.

• Minor Version (8 bits): Indicates minor version in use. For SSLv3, the value is 0.

• Compressed Length (16 bits): The length in bytes of the plaintext fragment (or compressed fragment if compression is used). The maximum value is $2^{14} + 2048$.

The content types that have been defined are change_cipher_spec, alert, handshake, and application_data.

SSL Change Cipher Spec Protocol

The Change Cipher Spec Protocol is one of the three SSL-specific protocols that use the SSL Record Protocol, and it is the simplest. This protocol consists of a single message, which consists of a single byte with the value 1.

The sole purpose of this message is to cause the pending st te to be copied into the current state, which updates the cipher suite to be used on th s connection.

SSL Alert Protocol

The Alert Protocol is used to convey SSL-r lat alerts to the peer entity. As with other applications that use SSL, alert messages are compressed and encrypted, as specified by the current state. Each me age in this protocol consists of two bytes.

The first byte takes the value warning(1) or fatal(2) to convey the severity of the message. If the level is fatal, SSL immediately terminates the connection. Other connections on the same session may continue, b t no new connections on this session may be established. The second byte contains a code that indicates the specific alert. The fatal alerts are listed below:

• unexpected_message: An inappropriate message was received.

• bad_record_mac: An incorrect MAC was received.

• decompression_failure: The decompression function received improper input (e.g., unable to decompress or decompress to greater than maximum allowable length).

• handshake_failure: Sender was unable to negotiate an acceptable set of security parameters given the options available.

• illegal_parameter: A field in a handshake message was out of range or inconsistent with other fields.

The remainder of the alerts are given below:

• close_notify: Notifies the recipient that the sender will not send any more messages on this connection. Each party is required to send a close_notify alert before closing the write side of a connection.

• no_certificate: May be sent in response to a certificate request if no appropriate certificate is available.

• bad_certificate: A received certificate was corrupt (e.g., contained a signature that did not verify).

• unsupported_certificate: The type of the received certificate is not supported.

• certificate_revoked: A certificate has been revoked by its signer.

• certificate_expired: A certificate has expired.

• certificate_unknown: Some other unspecified issue arose in processing the certificate, rendering it unacceptable.

SSL Handshake Protocol

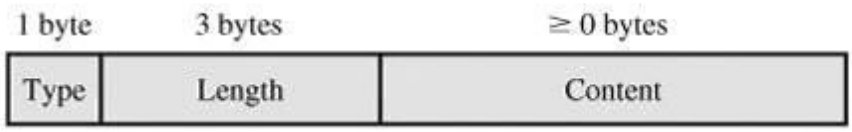
SSL Handshake protocol ensures establishment of reliable nd secure session between client and server and also allows server & client to:

• authenticate each other

• to negotiate encryption & MAC algorithms

• to negotiate cryptographic keys to be used

The Handshake Protocol consists of a eries of messages exchanged by client and server. All of these have the format shown below and each message has three fields:



(c) Handshake Protocol

• **Type (1 byte**): Indicates one of 10 messages.

• **Length (3 bytes):** The length of the message in bytes.

• **Content (>=0 bytes):** The parameters associated with this message

The following figure shows the initial exchange needed to establish a logical connection between client and server. The exchange can be viewed as having four phases.

o Establish Security Capabilities

o Server Authentication and Key Exchange o Client Authentication and Key Exchange

o Finish

Phase 1. Establish Security Capabilities

This phase is used to initiate a logical connection and to establish the security capabilities that will be associated with it. The exchange is initiated by the client, which sends a client_hello message with the following parameters:

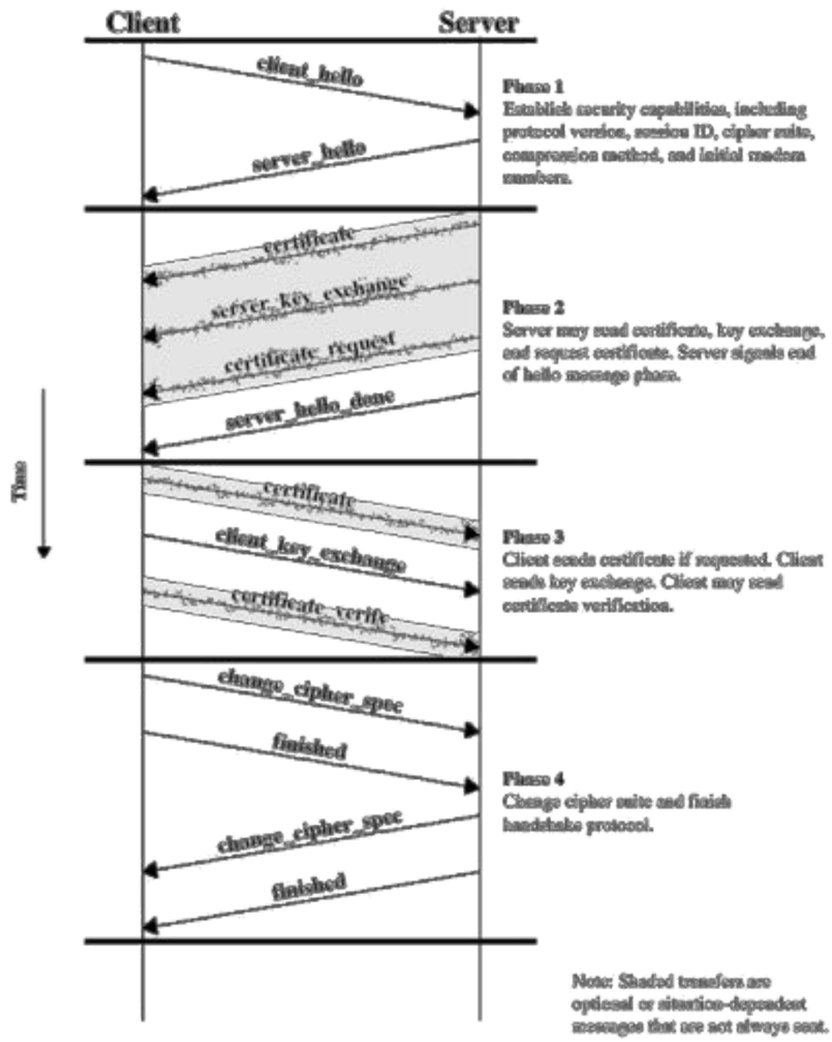
• Version: The highest SSL version understood by the client.

• Random: A client-generated random structure, consisting of a 32-bit timestamp and 28 bytes generated by a secure random number generator. These values serve as nonces and are used during key exchange to prevent replay attacks.

Session ID: A variable-length session identifier. A nonzero value indicates that the client wishes to update the parameters of an existing connection or create a new connection on this session. A zero value indicates that the cl ent wishes to establish a new connection on a new session.

• CipherSuite: This is a list that contains the combinations of cryptographic algorithms supported by the client, in decreasing order of preference. Each element of the list (each cipher suite) defines both a key exchange algorithm and a CipherSpec.

• Compression Method: This is a list of the compression methods the client supports.



Phase 2. Server Authentication and Key Exchange

The server begins this phase by sending its certificate via a certificate message, which contains one or a chain of X.509 certificates. The **certificate message** is required for any agreed-on key exchange method except anonymous Diffie-Hellman. Next, a **server_key_exchange** message may be sent if it is required. It is not required in two instances: (1) The server has sent a certificate with fixed Diffie-Hellman parameters, or (2) RSA key exchange is to be used.

Phase 3. Client Authentication and Key Exchange

Once the server_done message is received by client, it should verify whether a valid certificate is provided and check that the server_hello parameters are acceptable. If all is satisfactory, the client sends one or more messages back to the server. If the server has requested a certificate, the client begins this phase by sending a **certificate message**. If no suitable certificate is available, the client sends a no_certificate alert instead. Next is the **client_key_exchange** message, for which the content of the message depends on the type of key exchange.

Phase 4. Finish

This phase completes the setting up of a secure connection. The client sends a **change_cipher_spec** message and copies the pending CipherSpec into the current CipherSpec. The client then immediately sends the finished message under the new algorithms, keys, and secrets. The finished message verifies that the key exchange and authentication processes were successful.

The Purchase Response message includes a response block that acknowledges the order and references the corresponding transaction number. This block is signed by the merchant using its private signature key. The block and its signature are sent to the customer, along with the merchant's signature certificate. Necessary action will be taken by cardholder's software upon verification of the certificates and signature.