Secure Sockets Layer
Secure Sockets Layer

SSL/TLS provides endpoint authentication and communications privacy over the Internet using cryptography.

For web browsing, email, faxing, other data transmission.

In typical use, only the server is authenticated while the client remains unauthenticated.

Mutual authentication requires PKI deployment to clients.

Protocols allow client/server applications to communicate in a way designed to prevent eavesdropping, tampering, and message forgery.
Secure Sockets Layer

Presentation Layer:
Provides independence from differences in data representation among applications. OS functions translate data formats (app protocol syntax) to a uniform network format (bit stream to be transmitted), and vice versa so as to eliminate network compatibility problems. Takes care of encryption and data compression.
Secure Sockets Layer

Developed by Netscape Communications Corp (1995)
Ensures data privacy: transmission of data via encryption
Supports Server and Client authentication
Supports authentication of service via certificate
Ensures data integrity
Application independent - ftp, http, telnet are layered on top of it. Mainly used in https applications.
Can negotiate encryption keys
Sits on top of TCP/IP, does not require OS changes
Can be used to create a tunnel for VPN
Encryption and compression apply only to application layer
Secure Sockets Layer

Three phases:
1. Peer negotiation for algorithm support (see below)
2. Public key encryption based key exchange and certificate based authentication
3. Symmetric cipher based traffic encryption

Cryptographic choices:
for public-key cryptography: RSA, Diffie-Hellman, DSA, Fortezza;
for symmetric ciphers: RC2, RC4, IDEA, DES, Triple DES or AES;
for one-way hash functions: MD2, MD4, MD5 or SHA-1 (SHA-256 with TLS)

https://globalsign.ssllabs.com/analyze.html?viaform=on&d=
Secure Sockets Layer

Security features:

- Numbering all the records and using the sequence number in the Msg. Authen. Codes. (to prevent replay attacks)
- Uses HMAC for message integrity check
- Protection against several other known attacks
  - MIM attack involving a downgrade of the protocol to a less secure version via integrity check of initial handshake
  - identity fraud via digital signatures
  - known plaintext attacks via strong cryptographic algorithms
- The message that ends the handshake ("Finished") sends a hash of all the exchanged data in handshake (Integrity check to prevent Man in the Middle and Truncation attacks).
Secure Sockets Layer

Security features:

- A pseudorandom function splits the input data in half and processes each half with a different hashing algorithm, then XORs them together. Provides protection if one of these algorithms is found to be vulnerable.  
  
*TLS only*
Secure Sockets Layer

Incorrect uses are possible:

The form submission page is secured but not the login page
- serve the login page from https address, post the data to an http address

MS IE7, Old Google Chrome gave no warning
Firefox, Opera, Konqueror warn about this

Display of a secure page with non-secure media
Used to be revealing: https://java.com, http://www.java.com

Past Culprits:

Lots including J.C. Penny, Bank of America, J.P. Morgan, Oracle
Secure Sockets Layer

Rogue packet detection:

TCP cannot determine bogus data, sends it to SSL
SSL integrity checks that it is bogus and discards it
SSL cannot tell TCP to accept the real data
When real data follows, TCP rejects it due to a repeated sequence number
Since connection cannot be guaranteed, SSL has no choice but to close the connection
Secure Sockets Layer

Rogue packet detection:

TCP cannot determine bogus data, sends it to SSL
SSL integrity checks that it is bogus and discards it
SSL cannot tell TCP to accept the real data
When real data follows, TCP rejects it due to a repeated sequence number
Since connection cannot be guaranteed, SSL has no choice but to close the connection

This is considered to be a feature!
Secure Sockets Layer

SSL Attacker (DoS):

DoS attacks specific to SSL generally involve causing server overload

1. Since TCP handshake begins the connection request, Syn flooding is possible (but this is not specific to SSL)

2. After the TCP handshake, there is an SSL handshake which involves a lot of computation. If an attacker sends garbage, it is processed before closing the connection. Since TCP handshake is completed, firewalls cannot get involved: they think they are sending legitimate data

3. After the SSL handshake, the attacker can request renegotiation of the encryption method (if allowed). Then renegotiation is again requested and so on.
Secure Sockets Layer

IPSec vs. SSL for VPN:

- Some ISPs block IPSec traffic unless the customer pays - cannot do this for SSL since it's web-based and people buy, sell, manage bank accounts using it.
- IPSec requires client (OS) software but SSL is built into applications like web browsers.
- IPSec may have trouble with NATed routers, SSL doesn't
- Performance load of both is about the same since same encryption algorithms are used.
- Authentication in IPSec is both ways – maybe not for SSL.
- Some people want access to specific apps rather than subnets
Secure Sockets Layer

Uses several protocols organized into layers as follows:

<table>
<thead>
<tr>
<th>SSL handshake protocol</th>
<th>SSL cipher change protocol</th>
<th>SSL alert protocol</th>
<th>Application Protocol (eg. HTTP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL Record Protocol</td>
<td>TCP</td>
<td>IP</td>
<td></td>
</tr>
</tbody>
</table>

**SSL Record Protocol**: handles data security and integrity; encapsulates data sent by higher level protocols

**Handshake, Cipher change, Alert**: establish a connection; session management, crypto management, SSL message transfer
Secure Sockets Layer

Definitions:

Connection: logical 2-node peer-to-peer link – provides a service

Session: association between peers defining crypto algorithms, sequence numbers, etc. Created by handshake protocol. Used to avoid renegotiation of parameters from connection to connection

Session State:

- session identifier: generated by the receiver
- peer certificate: X.509 spec
- compression method: prior to encryption
- CipherSpec: encryption, integrity, and hash algorithms
- MasterSecret: 48 byte shared secret
Secure Sockets Layer

Definitions:

Connection State:
- Random numbers - chosen by server and client to make crypto breaking harder
- Server write MAC secret - used on data from server
- Client write MAC secret - used on data from client
- Server write secret key - server encryption, decryption by client
- Client write secret key - client encryption, decryption by server
- Initialization vectors - for CBC ciphers
- Sequence number - for both transmitted and received messages on both client and server sides
Secure Sockets Layer

SSL Record Protocol:

- Fragment the data that needs to be sent: create records
- Encapsulate them with appropriate headers
- Create an encrypted object that can be sent over TCP
Secure Sockets Layer

SSL Record Protocol:

- Fragment the data that needs to be sent: create records
- Encapsulate them with appropriate headers
- Create an encrypted object that can be sent over TCP

**Header of each record:** length of record and of data block

**Contents of record after header:** data, padding, MAC

\[
MAC = \text{hash \{secret key, data+padding, sequence number\}}
\]

where hash uses specified (negotiated) algorithm like SHA-1
Secure Sockets Layer

SSL Record Protocol:

Fragment the data that needs to be sent: create records
Encapsulate them with appropriate headers
Create an encrypted object that can be sent over TCP

Header of each record: length of record and of data block

Contents of record after header: data, padding, MAC
MAC = hash \{ secret key, data+padding, sequence number \}
where hash uses specified (negotiated) algorithm like SHA-1

Encrypted Object: encrypt record plus MAC

Header of EO: content-type: which of four protocols to use
to handle the data in the EO after decryption.
Protocol Major and Minor version numbers.
Secure Sockets Layer
Secure Sockets Layer

Cryptographic Doom Principle (CDP):

If you design a protocol that performs any cryptographic operation on a message before verifying the integrity of the message your protocol inevitably, somehow, is doomed

- Moxie Marlinspike
  
  former head of the security team at Twitter
  
  author of a proposed SSL replacement (Convergence)
  
  founder of Open Whisper Systems and
  
  co-author of the Signal Protocol for instant messaging
  
  member of the Institute for Disruptive Studies
Notes:
1. Padding may have to be added to the last block of plaintext
Secure Sockets Layer

Cryptographic Doom Principle (CDP)

Applied to SSL/TLS

Notes:
1. Padding may have to be added to the last block of plaintext
2. Value of each pad byte is the number of bytes being added so it is easy to check that padding is not valid
## Secure Sockets Layer

### Cryptographic Doom Principle (CDP)

### Applied to SSL/TLS

<table>
<thead>
<tr>
<th>m1</th>
<th>m2</th>
<th>m3</th>
<th>m4</th>
<th>m5</th>
<th>m6</th>
</tr>
</thead>
</table>

### Notes:
1. Padding may have to be added to the last block of plaintext.
2. Value of each pad byte is the number of bytes being added, so it is easy to check that padding is not valid.
3. SSL generates a padding error if the plaintext padding is not valid.
Secure Sockets Layer

Cryptographic Doom Principle (CDP)

Applied to SSL/TLS

<table>
<thead>
<tr>
<th>m1</th>
<th>m2</th>
<th>m3</th>
<th>m4</th>
<th>m5</th>
<th>m6</th>
</tr>
</thead>
</table>

IV → ⊕ → ⊕ → ⊕ → ⊕ → ⊕ → ⊕

Secret → D → D → D → D → D → D

<table>
<thead>
<tr>
<th>c1</th>
<th>c2</th>
<th>c3</th>
<th>c4</th>
<th>c5</th>
<th>c6</th>
</tr>
</thead>
</table>

Notes:
1. Padding may have to be added to the last block of plaintext
2. Value of each pad byte is the number of bytes being added

| End of plaintext | 0x05 | 0x05 | 0x05 | 0x05 | 0x05 |

so it is easy to check that padding is not valid
3. SSL generates padding error if the plaintext padding is not valid
4. SSL generates a MAC error if the integrity check fails
Secure Sockets Layer

Cryptographic Doom Principle (CDP)

Applied to SSL/TLS

<table>
<thead>
<tr>
<th>m1</th>
<th>m2</th>
<th>m3</th>
<th>m4</th>
<th>m5</th>
<th>m6</th>
</tr>
</thead>
</table>

Notes:
5. Attacker assumed to be able to control c5 entry to the xor
Secure Sockets Layer

Cryptographic Doom Principle (CDP)

Applied to SSL/TLS

Notes:
5. Attacker assumed to be able to control c5 entry to the xor
6. Attacker aims to decrypt c6 by crafting a c5
Secure Sockets Layer

Cryptographic Doom Principle (CDP)

Applied to SSL/TLS

Notes:
5. Attacker assumed to be able to control c5 entry to the xor
6. Attacker aims to decrypt c6 by crafting a c5
7. Attacker sets lowest byte of c5 to 0x00, other bytes random
   If padding error, Attacker sets lowest byte to 0x01, repeats
   until MAC error - last m6 byte is 0x01 since that would provide
   correct padding
Notes:
8. Let $P_n$ be value of last byte of modified $c5$ that gives MAC error then $I_n = P_n \oplus 0x01$ and $I_n = m6_n \oplus c5_n$
so $m6_n = c5_n \oplus P_n \oplus 0x01$
But Attacker knows $c5_n$, $P_n$ and therefore $m6_n$
9. Attacker then gets the next to last byte, 3$^{rd}$ to last ...
Secure Sockets Layer

Alert Protocol:

Used to transmit session messages. Each message:

<table>
<thead>
<tr>
<th>Fatal</th>
<th>Warning</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 byte</td>
<td></td>
<td>1 byte</td>
</tr>
</tbody>
</table>

Alert messages are compressed and encrypted according to the current session state

⇔ Errors occurring during handshake

⇐ Errors occurring during processing at server
Secure Sockets Layer

Change Cipher Spec Protocol:

Sent to close a pending session, setting in stone all the crypto parameters to be used in connections resulting from that session.

Change cipher messages are compressed and encrypted according to the current session state.

Each message:

```
1

1 byte```

Handshake Protocol:

To initiate a session: crypto negotiation

**Phase 1**: initiate (client_hello), identity not revealed!

Establish logical connection, negotiate session parms

---

**Version**: Highest SSL version supported by client

**R1**: random number

**SessionID**: non-0: resume earlier session, modify parms of this session, spawn an independent connection without handshake

0: initiate a session
Handshake Protocol:

To initiate a session: crypto negotiation

**phase 1**: initiate (client_hello), identity not revealed!
Establish logical connection, negotiate session parms

**Version**: Lowest SSL version supported by server

**R2**: a random number

**SessionID**: same as client's, if non-0, otherwise an ID decided by server

**CryptoAccept**: key exchange method, encrypt algorithm, hash function
Handshake Protocol:

To initiate a session: crypto negotiation

**phase 2**: authenticate (server_hello)

---

Certificate(s): chain of certificates to a trusted CA to authenticate server

**Diffie-Hellman value**: optional

**Request for certificate from client**: optional

**server_done**: completes the message sequence
Secure Sockets Layer

Handshake Protocol:

To initiate a session: crypto negotiation

phase 2: authenticate (server_hello)

Client verifies certificate, checks date and invalidation lists
Client checks that the certifying authority is trusted
Client checks the CA's public key against that of the certificate
Client checks that the domain name in the certificate matches that of server
How the client authenticates the server
Secure Sockets Layer

Handshake Protocol:

To initiate a session: too many secrets

Client chooses random number $S$ (pre-master secret) and encrypts it with server's public key (server\{S\})
Client computes master key as $K=f(S,R1,R2)$, computes hash($K+msgs$)
Six secret keys - 3 from client to server and 3 from server to client
integrity, encryption, initialization vector (derived from $S$)
Handshake Protocol:

To initiate a session: crypto negotiation

**phase 3**: start key exchange

S sent encrypted by server's public key, server computes master secret

**Key exchange msg**: delivers the keys, depends on the agreed key exchange method

**Send certificate of client**: if requested by server
Handshake Protocol:

To initiate a session: crypto negotiation
phase 4: confirmation and setup

Change cipher spec msg: crypto spec now considered agreed
Setup of algorithms: make cryptosystems ready to go
Finished msg: encrypted with agreed upon crypto algorithms and keys so server can verify that communication is possible. Is the hash of all the messages in the handshake.
Handshake Protocol:

To initiate a session: crypto negotiation

**phase 4**: confirmation and setup

Same finished message is sent back to client.
Session is terminated and the TCP connection is closed but the "state" of the session is saved to be reopened later with same parameters.
How the server authenticates the client
The Main Usage of SSL

- Secure "tunnel" through the Internet
- Non-Internet (telephone) line
- Consumer must trust merchant with card
- Similar to ordinary phone order
- High transaction costs

Consumer

Internet

Merchant

Credit Card Acquirer

Issuer

Issuer bills Consumer

Acquirer notifies Issuer

Credit Card Issuer
Secure Sockets Layer

Session Resumption:

Per-session master key is established using expensive public key cryptography

Connections cheaply derived from master key with handshake involving nonces, not public keys

SessionID and master key for the session is stored by server to support resumption

If server loses the state, it can be reestablished by the client sending S encrypted with server's public key
Secure Sockets Layer

Handshake Protocol (Resumption): Using Session-ID

Client

sessionID, ciphers, R1

Server
Secure Sockets Layer

Handshake Protocol (Resumption): Using Session-ID

sessionID, cipher, R2, hash(S,R1,R2)
Secure Sockets Layer

Handshake Protocol (Resumption): Using Session-ID

Client

ChangeCipherSpec
what follows is encrypted

Server
Secure Sockets Layer

Handshake Protocol (Resumption): Forgot Session-ID

ciphers, R1
Handshake Protocol (Resumption): Forgot Session-ID

Client

Server

sessionID, cipher, R2, certificate
Secure Sockets Layer

Handshake Protocol (Resumption): Forgot Session-ID

Client

Server\{S\}, K\{hash(S,R1,R2)\}

Server
Handshake Protocol (Resumption): Forgot Session-ID

\[ \text{hash}(S, R1, R2) \]
Secure Sockets Layer

Computing Keys:

If Diffie-Hellman is used to compute pre-master Secret S, fixed DH is allowed! Same session key repeated!! Ephemeral DH, ephemeral elliptic curve DH are also allowed for perfect forward secrecy (e.g. gmail)

If kept around in memory, a fixed DH key could compromise future communications

So it is hashed with nonces to get the master secret (Stealing master secret only affects this communication)
Transport Layer Security

TLS vs. SSL:

SSL connections begin with security and proceed directly to secured communications.

TLS connections begin with an insecure “hello” to the server and only switch to secured communications after the handshake between the client and the server is successful.

TLS is an open community standard and is therefore more extensible.

TLS allows flexibility in choosing ciphers of varying complexities – hence is non-interoperable with SSL.

TLS is no longer backward compatible with SSL – will never negotiate lower than SSL 3.0.
TLS Vulnerabilities:

Drown Attack – still affects high percentage of servers:
Configuration exploit – attacker forces a server to use SSL v.2 which is known to be badly insecure.

No modern clients use SSL v.2 – some servers have kept supporting SSL v.2 because it did not seem to matter. Configuration change would prevent this attack.

SSL v.2 vulnerable to Ciphersuite rollback attack (MiM)
SSL v.2 allows DES encryption (cracking in a few days)
Signature Verification Failed vulnerability – client may continue with the connection without authentication
SSL v.1 doesn't block disabled ciphers
Divide and conquer session key recovery in SSL v.2
DH primes that are not safe are allowed – can be recovered if static DH is used
Transport Layer Security

TLS Vulnerabilities:

Key Compromise Impersonation vulnerability:
Attacker can impersonate trusted servers without being in possession of the servers' secret keys

Attacker spoofs server, forces client to use insecure TLS authentication options and a certificate for which attacker has the private key

Victim gets attacker's certificate through social engineering e.g. attacker presents victim with certificate to use a service

Uses the fact that many operations manage certificates insecurely
Transport Layer Security

TLS Vulnerabilities:

Browser Exploit Against SSL/TLS (BEAST):
Attacker can determine an initialization vector

If IVs are known, an attacker can get information from ciphertext patterns, although not a lot

Requires another vulnerability such as XXS

Factoring RSA Keys (FREAK) & Logjam:
Reduce the security offered by SSL/TLS by forcing a connection to use “Export-grade” grade encryption
- which reduces the RSA strength to 512 bits

Numerous Occurrence Monitoring & Recovery Exploit:
Attack against RC4 which allows a HTTP cookie to be retrieved in a couple of days
Transport Layer Security

TLS Vulnerabilities:

**Bar Mitzvah:**
Small amounts of plaintext can be recovered from streams protected by RC4 encryption

**Sweet32:**
Capture a HTTP cookie after grabbing 785 GB of traffic

**TLS Padding Oracle On Downgraded Legacy Encryption (POODLE) vulnerability:**
Possible when a block cipher is enabled utilizing the CBC cipher mode
Allows an attacker to decipher a chosen byte of cipher text in as few as 256 attempts.

**Heartbleed:**
Theft of credentials – buffer overrun
Transport Layer Security

TLS Certificates:

Self-Signed:
Attacker can create its own “forged” certificate
End user has no way of knowing the certificate is bogus
Attacker can capture all encrypted data and modify client requests and server responses.

Certificate with wrong hostname:
Certificates confirm the identity of a service
The identity is specified by the Common Name (CN)
CN != hostname raises security error in browsers
Attacker now has the means to place illegitimate cert on end user's machine – the end user clicks OK to accepting the certificate

Try it: https://globalsign.ssllabs.com/analyze.html?d=