Secure Sockets Layer (SSL) and Man-in-the-Middle Vulnerability

Prof. Ravi Sandhu
Executive Director and Endowed Chair

Lecture 5

ravi.utsa@gmail.com
www.profsandhu.com
Internet Security Protocols
TCP RFC 793
Sept. 1981
IPv4 RFC 791
Sept. 1981
Internet Security Protocols

TCP RFC 793
Sept. 1981
IPv4 RFC 791
Sept. 1981

Where to inject security?
Internet Security Protocols

- TCP RFC 793, Sept. 1981
- HTTP, FTP, SMTP
- TCP, UDP
- IP
- Data link layer protocols
- Physical layer protocols
- SET, 1996
- SSL, 1994
- IPsec, 1998
Internet Security Protocols

Dozens of other security protocols

Some successes
Many failures

Half successful
SSL, 1994
IPsec, 1998
Largely failed

© Ravi Sandhu
1-way vs 2-way SSL

**1-way SSL**

Client (Browser) → 1-way SSL → Server

**2-way SSL**

Client (Browser) → 2-way SSL → Server
1-way vs 2-way SSL

1-way SSL

1-way SSL

2-way SSL

2-way SSL

Client (Browser) 1-way SSL Server

Client (Browser) 2-way SSL Server

RSA encryption certificate

RSA encryption certificate

RSA signature certificate

RSA encryption certificate

RSA signature certificate
1-way vs 2-way SSL

1-way SSL

Client (Browser) ➔ Server

LESS SECURE
Phishing
Man-in-the-middle

2-way SSL

Client (Browser) ➔ Server

MORE SECURE
Phishing
Man-in-the-middle
1-way vs 2-way SSL

1-way SSL

Client (Browser) ➔ Server

LESS SECURE
Phishing
Man-in-the-middle

MASS DEPLOYMENT

2-way SSL

Client (Browser) ➔ Server

MORE SECURE
Phishing
Man-in-the-middle

MINIMAL DEPLOYMENT
The SSL Lesson

- Client-less trumps client-full
- Start-ups (SSL) trump committees (IPSEC)
SSL Details
SSL

- layered on top of TCP
- SSL versions 1.0, 2.0, 3.0, 3.1
- Netscape protocol
- later refitted as IETF standard TLS (Transport Layer Security)
- TLS 1.0 very close to SSL 3.1
application protocol independent
does not specify how application protocols add security with SSL
  - how to initiate SSL handshaking
  - how to interpret certificates
left to designers of upper layer protocols to figure out
SSL vs TCP Ports

- https 443
- ssmtp 465
- snntp 563
- sldap 636
- spop3 995
- ftp-data 889
- ftsp 990
- imaps 991
- telnets 992
- ircs 993
SSL Services

- peer entity authentication
- data confidentiality
- data authentication and integrity
- compression/decompression
- generation/distribution of session keys
  - integrated into protocol
- security parameter negotiation
## SSL Architecture

<table>
<thead>
<tr>
<th></th>
<th>SSL Change Cipher Spec Protocol</th>
<th>SSL Alert Protocol</th>
<th>HTTP</th>
<th>Other Application Protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL Handshake Protocol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### SSL Record Protocol

**TCP**

**IP**
Handshake protocol: complicated
- embodies key exchange & authentication
- runs in plaintext
- 10 message types

Change Cipher Spec protocol: straightforward
- single 1 byte message with value 1
- could be considered part of handshake protocol
- transitions from plaintext to encrypted and mac’ed

Record protocol: straightforward
- fragment, compress, MAC, encrypt
- uses 4 symmetric keys

Alert protocol: straightforward
- 2 byte messages
- 1 byte alert level- fatal or warning; 1 byte alert code
SSL Record Protocol

- 4 symmetric keys

Client (Browser)

Key 1 for MAC
Key 2 for encrypt

Server

Key 3 for MAC
Key 4 for encrypt
SSL Record Protocol

- 4 steps by sender (reversed by receiver)
  - Fragmentation
  - Compression
  - MAC
  - Encryption
SSL Record Protocol

- each SSL record contains
  - content type: 8 bits, only 4 defined
    - change_cipher_spec
    - alert
    - handshake
    - application_data
  - protocol version number: 8 bits major, 8 bits minor
  - length: max 16K bytes (actually $2^{14}+2048$)
  - data payload: optionally compressed and encrypted
  - message authentication code (MAC)
initially SSL session has null compression and cipher algorithms
both are set by the handshake protocol at beginning of session
handshake protocol may be repeated during the session
SSL Session

- SSL session negotiated by handshake protocol
  - session ID
    - chosen by server
  - X.509 public-key certificate of peer
    - possibly null
  - compression algorithm
  - cipher spec
    - encryption algorithm
    - message digest algorithm
  - master secret
    - 48 byte shared secret
  - is resumable flag
    - can be used to initiate new connections
    - each session is created with one connection, but additional connections within the session can be further created
SSL Connection State

- connection end: client or server
- client and server random: 32 bytes each
- keys generated from master secret, client/server random
  - client_write_MAC_secret  server_write_MAC_secret
  - client_write_key  server_write_key
  - client_write_IV  server_write_IV
- compression state
- cipher state: initially IV, subsequently next feedback block
- sequence number: starts at 0, max $2^{64}-1$
SSL Connection State

- 4 parts to state
  - current read state
  - current write state
  - pending read state
  - pending write state
- handshake protocol
  - initially current state is empty
  - either pending state can be made current and reinitialized to empty
SSL Handshake Protocol

- Type: 1 byte
  - 10 message types defined
- length: 3 bytes
- content
SSL Handshake Protocol

Fig. 1 - Message flow for a full handshake

* Indicates optional or situation-dependent messages that are not always sent.
SSL Handshake Protocol

- **Phase 1:**
  - Establish security capabilities

- **Phase 2:**
  - Server authentication and key exchange

- **Phase 3:**
  - Client authentication and key exchange

- **Phase 4:**
  - Finish
这些握手消息必须按照顺序发生。

- 可选消息可以被删除。
- 第10条消息
  - `hello_request`
  - 可以在任何时候从服务器发送到客户端，以请求客户端开始握手协议来重新协商会话。
- `change_cipher_spec` 是一个单独的1消息协议
  - 功能上就像握手协议中的消息一样。

- 该协议可以用于重新协商会话。
SSL 1-Way Handshake with RSA

* Indicates optional or situation-dependent messages that are not always sent.

Fig. 1 - Message flow for a full handshake
- Establish security capabilities
- client hello message
  - 4 byte timestamp, 28 byte random value
  - session ID:
    - non-zero for new connection on existing session
    - zero for new connection on new session
  - client version: highest version
  - cipher_suite list: ordered list
    - key exchange method, encryption method, MAC method
  - compression list: ordered list
- server hello message
  - 32 byte random value
  - session ID:
    - new or reuse
  - version
    - lower of client suggested and highest supported
  - cipher_suite list: single choice
  - compression list: single choice
SSL 1-Way Handshake with RSA

Fig. 1 - Message flow for a full handshake

* Indicates optional or situation-dependent messages that are not always always sent.
Server authentication and key exchange

- certificate message
  - server’s X.509v3 certificate followed by optional chain of certificates
  - required for RSA

- server done message
  - ends phase 2, always required
SSL 1-Way Handshake with RSA

**Phase 1**

Client

ClientHello

--------->

Server

ServerHello

Certificate*

CertificateRequest*

<-------

ServerHelloDone

**Phase 2**

Certificate*

ServerKeyExchange*

CertificateRequest*

<-------

ServerHelloDone

**Phase 3**

ClientKeyExchange

CertificateVerify*

[ChangeCipherSpec]

Finished

--------->

[ChangeCipherSpec]

Finished

**Phase 4**

Application Data

<-------->

Application Data

---

Fig. 1 - Message flow for a full handshake

* Indicates optional or situation-dependent messages that are not always sent.
SSL 1-way Handshake Phase 3

- Client authentication and key exchange
- client key exchange message
  - client generates 48-byte pre-master secret, encrypts with server’s RSA public key
- client and server compute 48 byte master secret
  - using 48-byte pre-master secret, ClientHello.random, ServerHello.random
- client and server compute 4 symmetric keys from master secret

Key 1 for MAC
Key 2 for encrypt
Key 3 for MAC
Key 4 for encrypt
SSL 1-Way Handshake with RSA

Phase 1

Client

ClientHello

--------> Server

Server

ServerHello

Certificate*

CertificateRequest*

<--------

Phase 2

Certificate*

ServerKeyExchange*

<-------- ServerHelloDone

Phase 3

Certificate*

ClientKeyExchange

CertificateVerify*

[ChangeCipherSpec]

Finished

--------> Server

[ChangeCipherSpec]

Finished

Phase 4

<-------- Application Data

<-------->

Application Data

Record Protocol

Fig. 1 - Message flow for a full handshake

* Indicates optional or situation-dependent messages that are not always sent.
- Finish and move to record protocol
- change cipher spec message
  - not considered part of handshake protocol but in some sense is part of it
  - 1 byte message protected by current state
  - copies pending state to current state
- Finished message
  - sent under new algorithms and keys
  - content is MAC of all previous messages with master secret and constant “client finished” or “server finished”
SSL 1-Way Handshake with RSA

Phase 1

Client → Server

ClientHello

Phase 2

Certificate* ← Certificate*  
ServerKeyExchange* 
CertificateRequest*  
ServerHelloDone

Phase 3

Certificate*

ClientKeyExchange

CertificateVerify*

[ChangeCipherSpec]

Finished →

Phase 4

[ChangeCipherSpec]

Finished ←

Application Data

Application Data

Record Protocol

* Indicates optional or situation-dependent messages that are not always always sent.

Fig. 1 - Message flow for a full handshake
### SSL 2-Way Handshake with RSA

<table>
<thead>
<tr>
<th>Phase 1</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Client</strong></td>
<td><strong>Server</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ClientHello</td>
<td>--------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>ServerHello</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Certificate*</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>ServerKeyExchange</strong></td>
<td><strong>CertificateRequest</strong>*</td>
<td>&lt;---------&gt;</td>
</tr>
<tr>
<td></td>
<td><strong>ServerHelloDone</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase 2</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Certificate</strong>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ServerKeyExchange</strong>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CertificateRequest</strong>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>ServerHelloDone</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase 3</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Certificate</strong>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ClientKeyExchange</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CertificateVerify</strong>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ChangeCipherSpec]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Finished</strong></td>
<td>--------&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase 4</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>[ChangeCipherSpec]</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Finished</strong></td>
<td>&lt;---------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Application Data</strong></td>
<td>&lt;--------&gt;</td>
<td><strong>Application Data</strong></td>
<td>&lt;--------&gt;</td>
</tr>
</tbody>
</table>

*Fig. 1 – Message flow for a full handshake*

* Indicates optional or situation-dependent messages that are not always sent.*
Server authentication and key exchange

certificate message
  - server’s X.509v3 certificate followed by optional chain of certificates
  - required for RSA

certificate request message
  - request a certificate from client
  - specifies Certificate Type and Certificate Authorities

server done message
  - ends phase 2, always required
SSL 2-way Handshake Phase 3

- Client authentication and key exchange
  - certificate message
    - client’s X.509v3 certificate followed by optional chain of certificates
  - client key exchange message
    - client generates 48-byte pre-master secret, encrypts with server’s RSA public key
  - certificate verify message
    - signs hash of master secret (established by key exchange) and all handshake messages so far

- client and server compute 48 byte master secret
  - using 48-byte pre-master secret, ClientHello.random, ServerHello.random
- client and server compute 4 symmetric keys from master secret
2 byte alert messages

- 1 byte level
  - fatal or warning
- 1 byte
  - alert code
SSL Alert Messages

Warning or fatal

close_notify(0),
unexpected_message(10),
bad_record_mac(20),
decryption_failed(21),
record_overflow(22),
decompression_failure(30),
handshake_failure(40),
bad_certificate(42),
unsupported_certificate(43),
certificate_revoked(44),
certificate_expired(45),
certificate_unknown(46),
illegal_parameter(47),
unknown_ca(48),
access_denied(49),
deencode_error(50),
decrypt_error(51),
export_restriction(60),
protocol_version(70),
insufficient_security(71),
internal_error(80),
user_canceled(90),
no_renegotiation(100),
SSL Alert Messages

- always fatal
  - unexpected_message
  - bad_record_mac
  - decompression_failure
  - handshake_failure
  - illegal_parameter
SSL Man-in-the-Middle (MITM) Attack
1-way SSL MITM

Client (Browser) → Server

https

RSA encryption certificate
SSL Lock Icon Evolution by Browser

IE
v5,6 v7,8 v9
Firefox:
v2 v3.4 osx v3.4 win v3.4 linux
Chrome:
Safari:
OSX win
Opera:
Konqueror:

http://elie.im/blog/
1-way SSL MITM

Client (Browser) → MITM → Server

http → https

RSA encryption certificate
1-way SSL MITM

Client (Browser) → MITM → Server

https  https

RSA encryption certificate
1-way SSL MITM

Client (Browser) -> MITM -> Server

https -> MITM

MITM -> https

fake server certificate

RSA encryption certificate
Server-Side Masquerading

Bob
Web browser

1 way SSL

www.host.com
Web server

Ultratrust
Security
Services

www.host.com
Server-Side Masquerading

Bob
Web browser

1-way SSL

Mallory’s
Web server

1-way SSL

www.host.com
Web server

Ultraprutz
Security
Services

www.host.com

BIMM
Corporation

www.host.com

© Ravi Sandhu
Server-Side Masquerading

Bob
Web browser

1-way SSL

BIMM Corporation

1-way SSL

Ultratrust Security Services

www.host.com

Mallory’s Web server

www.host.com

www.host.com

Ultratrust Security Services

www.host.com

© Ravi Sandhu

World-Leading Research with Real-World Impact!
1-way SSL MITM

Client (Browser)  ➔ MITM  ➔ Server

- https
- RSA encryption certificate
- fake server certificate
- RSA signature certificate
- fake client certificate
OpenSSL Heartbleed Attack

**How the Heartbleed Bug Works:**

1. **Server:** Are you still there? If so, reply “YES” (4 bytes).
   - *Request:
     - Server: Are you still there? If so, reply "YES" (4 bytes).
   - *Response:
     - Server: Yes.

2. **Server:** Are you still there? If so, reply "NO" (4 bytes).
   - *Request:
     - Server: Are you still there? If so, reply "NO" (4 bytes).
   - *Response:
     - Server: No.

3. **Client:** Are you still there? If so, reply "YES" (4 bytes).
   - *Request:
     - Client: Are you still there? If so, reply "YES" (4 bytes).
   - *Response:
     - Client: Yes.

4. **Client:** Are you still there? If so, reply "NO" (4 bytes).
   - *Request:
     - Client: Are you still there? If so, reply "NO" (4 bytes).
   - *Response:
     - Client: No.

5. **Client:** Are you still there? If so, reply "WAIT" (500 bytes).
   - *Request:
     - Client: Are you still there? If so, reply "WAIT" (500 bytes).
   - *Response:
     - Server: Yes.
     - Server: No.

6. **Client:** Are you still there? If so, reply "DIE" (500 bytes).
   - *Request:
     - Client: Are you still there? If so, reply "DIE" (500 bytes).
   - *Response:
     - Server: Yes.
     - Server: No.

7. **Server:** Are you still there? If so, reply "YES" (4 bytes).
   - *Request:
     - Server: Are you still there? If so, reply "YES" (4 bytes).
   - *Response:
     - Server: Yes.

8. **Server:** Are you still there? If so, reply "NO" (4 bytes).
   - *Request:
     - Server: Are you still there? If so, reply "NO" (4 bytes).
   - *Response:
     - Server: No.

9. **Server:** Are you still there? If so, reply "WAIT" (500 bytes).
   - *Request:
     - Server: Are you still there? If so, reply "WAIT" (500 bytes).
   - *Response:
     - Server: Yes.
     - Server: No.

10. **Server:** Are you still there? If so, reply "DIE" (500 bytes).
    - *Request:
      - Server: Are you still there? If so, reply "DIE" (500 bytes).
    - *Response:
      - Server: Yes.
      - Server: No.