Chapter 4: Securing TCP connections

Chapter goals:

- security in practice:
  - Security in the transport layer (versus other layers)
  - SSL / TLS
Chapter Roadmap

- Security in the transport layer
- SSL - The big picture
- SSL - A more complete picture

SSL: Secure Sockets Layer

- Widely deployed security protocol
  - Supported by almost all browsers and web servers
  - HTTPS
  - Billions $/year over SSL
- Mechanisms: [Woo 1994], Implementation: Netscape
- Number of variations:
  - TLS: transport layer security, RFC 2246
- Provides
  - Confidentiality
  - Integrity
  - Authentication
- Original goals:
  - Had Web e-commerce transactions in mind
  - Encryption (especially credit-card numbers)
  - Web-server authentication
  - Optional client authentication
  - Minimum hassle in doing business with new merchant
- Available to all TCP applications
  - Secure socket interface
SSL and TCP/IP

<table>
<thead>
<tr>
<th>Application</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>SSL</td>
</tr>
<tr>
<td>IP</td>
<td>TCP</td>
</tr>
</tbody>
</table>

Normal Application

Application with SSL

- SSL provides application programming interface (API) to applications
- C, Java, ... SSL libraries/classes readily available

Relative Location of Security Facilities in the TCP/IP Stack

<table>
<thead>
<tr>
<th>HTTP</th>
<th>FTP</th>
<th>SMTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL / TLS</td>
<td>TCP</td>
<td>IP</td>
</tr>
<tr>
<td>TCP</td>
<td>IP</td>
<td>IPsec</td>
</tr>
</tbody>
</table>

Security at transport level

| Security at network level |

- Both are general-purpose (i.e. application independent) solutions
- But
  - SSL is specific to TCP
    - Does not work with UDP, contrary to IPsec
    - But makes SSL simpler (no worry about loss and retransmission of data)
  - SSL only protects the TCP payload
    - Traffic analysis is thus possible
Secure Socket Layer - SSL

- SSLv2 was originated by Netscape in 1995
- Later came SSLv3
- IETF then introduced the TLS (Transport Layer Security) standard
  - Kind of SSLv3.1
    - Only minor changes, but not interoperable
      - Session key made stronger (harder to cryptanalyse)
    - More secure
- SSL/TLS is designed to make use of TCP to provide a reliable end-to-end secure service

Chapter Roadmap

- Security in the transport layer
- SSL - The big picture
- SSL - A more complete picture
Could do something like PGP:

But want to send byte streams & interactive data
Want a set of secret keys for the entire connection
Want certificate exchange part of protocol: handshake phase

“Toy SSL”: a simple secure channel

- **Handshake**: Alice and Bob use their certificates and private keys to authenticate each other and exchange shared secret
- **Key Derivation**: Alice and Bob use shared secret to derive set of keys
- **Data Transfer**: Data to be transferred is broken up into a series of records
- **Connection Closure**: Special messages to securely close connection
**Toy SSL: three phases**

1. **Handshake:**
   - Alice establishes TCP connection to Bob
   - authenticates Bob via CA signed certificate
   - creates, encrypts (using Bob’s public key), sends pre-master secret key to Bob
     - nonce exchange not shown

2. **Key Derivation:**
   - Considered bad to use same key for more than one cryptographic operation
     - Use different keys for message authentication code (MAC) and encryption
   - 4 keys:
     - $E_B$: Bob→Alice data encryption key
     - $E_A$: Alice→Bob data encryption key
     - $M_B$: Bob→Alice MAC key
     - $M_A$: Alice→Bob MAC key
   - Keys derived from key derivation function (KDF)
     - Takes master secret and (possibly) some additional random data and creates the keys
Toy SSL: three phases

- Why not encrypt data in constant stream as we write it to TCP?
  - Where would we put the MAC? If at end, no message integrity until all data processed
  - For example, with instant messaging, how can we do integrity check over all bytes sent before displaying?
- Instead, break stream in series of records
  - Each record carries a MAC
  - Receiver can act on each record as it arrives
- Issue: in record, receiver needs to distinguish MAC from data
  - Want to use variable-length records

<table>
<thead>
<tr>
<th>length</th>
<th>data</th>
<th>MAC</th>
</tr>
</thead>
</table>

3. Data transfer

Toy SSL data transfer

Useful to ensure integrity of the whole byte stream.
seq # not sent, just used in the hash

TCP byte stream. Compression can take place
block n bytes together

TCP byte stream. Compression can take place
block n bytes together

compute MAC
encrypt d + MAC

SSL record format

Type Ver Len | d | H(d) |
|------------|---|------|

Useful to ensure integrity of the whole byte stream.
seq # not sent, just used in the hash
Sequence Numbers

Problem: Attacker can capture and replay record or re-order records
Solution: Put sequence number into MAC:
- $MAC = MAC(M_x, \text{sequence}||\text{data})$
- Note: no sequence number field

Problem: Attacker could still replay all records
Solution: Add nonces in handshake, and use them to build keys, e.g. $M_x$
- see detailed protocol

Control information

Problem: Truncation attack:
- Attacker forges TCP connection close segment
- One or both sides thinks there is less data than there actually is
Solution: record types, with one type for closure
- type 0 for data; type 1 for closure
$MAC = MAC(M_x, \text{sequence}||\text{type}||\text{data})$
Toy SSL: summary

Toy SSL isn’t complete

- How long are fields?
- Which encryption protocols?
- Want negotiation?
  - Allow client and server to support different encryption algorithms
  - Allow client and server to choose together specific algorithm before data transfer
Secure Socket Layer - SSL

SSL is composed of 2 sublayers
- The lower layer is the SSL Record Protocol:
  - Provides Integrity and Confidentiality
- The main protocol of the upper layer is the SSL Handshake Protocol, that we will study in more detail

<table>
<thead>
<tr>
<th>SSL Handshake Protocol</th>
<th>SSL Change Cipher Spec Protocol</th>
<th>SSL Alert Protocol</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>SSL Record Protocol</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TCP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IP</td>
</tr>
</tbody>
</table>
SSL Record Protocol

record header: content type; version; length

MAC: also includes sequence number, type and MAC key $M_x$

Fragment: each SSL fragment $2^{14}$ bytes (~16 Kbytes)

SSL Record Format

<table>
<thead>
<tr>
<th>1 byte</th>
<th>2 bytes</th>
<th>3 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>content type</td>
<td>SSL version</td>
<td>length</td>
</tr>
</tbody>
</table>

data

MAC

Data and MAC encrypted (symmetric algo)
SSL Handshake Protocol

- The most complex part of SSL
- Purpose
  1. Server authentication
  2. Negotiation: agree on crypto algorithms
  3. Establish keys
  4. Client authentication (optional)
- This mechanism is called session creation
  - a session defines the set of cryptographic security parameters to be used
  - multiple secure TCP connections between a client and a server can share the same session
    - less computation cost
- This handshake protocol is used before any application data is transmitted

The four phases of the SSL Handshake Protocol

1. Establish Security Capabilities
2. Server Authentication (and key exchange)
3. (Client Authentication and) key exchange
4. Finish
Phase 1 of SSL Handshake: Establish Security Capabilities

Client_hello contains the combinations of cryptographic algorithms supported by the client, in decreasing order of preference.

Server_hello is the selection by the server:
- Assign a session_id
- Select the CipherSpec

The client_hello can contain a session_id:
- To resume a previous session

Both messages have nonces.

Why the two random nonces?

- Suppose Trudy sniffs all messages between Alice & Bob
- Next day, Trudy sets up TCP connection with Bob, sends the exact same sequence of records:
  - Bob (Amazon) thinks Alice made two separate orders for the same thing
  - Solution: Bob sends different random nonce for each connection. This causes encryption and MAC keys to be different on the two days
  - Trudy’s messages will fail Bob’s integrity check
CipherSpecs

- The CipherSpec contains fields like:
  - Cipher Algorithm (DES, 3DES, RC4, AES)
  - MAC Algorithm (based on MD5, SHA-1)
  - Public-key algorithm (RSA)

Phase 2 of SSL Handshake:
Server Authentication (and Key Exchange)

We first consider the most classical key exchange protocol: RSA

- Certificate
  - RSA certificate
  - Client software comes configured with public keys of various "trusted" organizations (CAs) to check certificate (chains)
    - Security threat!
- (Server_key_exchange)
  - With RSA: not used
- Certificate_request
  - Server may request client certificate
  - Usually not done
- Server_hello_done

All these messages are usually combined with the previous server_hello message.
Phase 3 of SSL Handshake: 
(Client Authentication and) Key Exchange

- **Certificate**
  - Only if requested by server
- **Client_key_exchange**
  - For RSA: It’s the pre-master secret (PMS) encrypted with the server’s public key
  - Also sent is a hash of \((\text{PMS}, \text{R}_A, \text{R}_B)\)
- **Certificate_verify**
  - If certificate sent by client
  - Used by the client to prove he has the private key associated with its certificate (in case someone is misusing the client’s certificate)
  - Basically, the client signs a hash of the previous messages

Phase 4 of SSL: Handshake Finish

- **Change_cipher_spec**
  - Its purpose is to cause the pending state to be copied into the current state
  - From now on, all records are encrypted and integrity-protected
  - Is part of the Change Cipher Spec protocol
- **Finished**
  - Verifies that the key exchange and authentication processes were successful
  - It is the concatenation of 2 MAC values calculated from the previous messages
Role of the Finish phase

- **Counter the downgrade attack:**
  - An attacker could have removed the cipher suites with strong encryption from the client_hello message, causing the entities to agree upon a weaker cipher.

- **Counter the truncation attack:**
  - An attacker could close the underlying connection (by sending a TCP close message) which, in SSLv2, would have terminated the SSL session abnormally.

- **Note:** The truncation attack can also occur later during the data transfer:
  - The solution is to indicate in the type field of the SSL record whether it is the last record.
  - In normal operations, the TCP connection cannot be closed before these type fields have been exchanged over the SSL connection.

---

```
Real Connection

After TCP SYN and SYNACK
  ▼
  handshake: ClientHello
  ▼
  handshake: ServerHello
  ▼
  handshake: Certificate
  ▼
  handshake: ServerHelloDone
  ▼
  handshake: ClientKeyExchange
  ▼
  ChangeCipherSpec
  ▼
  handshake: Finished
  ▼
  ChangeCipherSpec
  ▼
  handshake: Finished
  ▼
  application_data
  ▼
  application_data
  ▼
  Alert: warning, close_notify

Everything henceforth is encrypted

TCP FIN follows
```
Main Key exchange methods based on RSA

- **RSA**
  - The classical method shown in previous slides
  - Client sends a PMS encrypted with the server’s certified RSA public key: $K_B(PMS)$
    - Server needs a certified encryption public key

- **RSA with signature-only key**
  - Encryption with a RSA key longer than 512 bits was not exportable
  - Server first generates a temporary pair of RSA (short) keys ($k_B, k_B^{-1}$) and sends the public one to the client, signed by its RSA (long-term) key: $K_B(k_B^{-1})$
  - Client sends a PMS encrypted with the server’s temporary RSA public key: $k_B(PMS)$
  - Note: this scheme designed for exportability actually enhances security because it allows (weak-key) perfect forward secrecy:
    - Breaking or stealing the temporary private key does not allow Trudy to decrypt previous SSL connections

Phases 2 and 3: RSA with “signature-only”

- **Server_key_exchange**
  - With RSA “signature-only”: it is the temporary public key, signed by the server’s long-term key

- **Client_key_exchange**
  - It’s the PMS encrypted with the server’s temporary public key
    - $+ \text{ hash of } (PMS, R_A, R_B)$

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>certificate</td>
<td>server_hello_done</td>
</tr>
<tr>
<td>server_key_exchange</td>
<td>certificate</td>
</tr>
<tr>
<td>certificate_request</td>
<td>server_key_exchange</td>
</tr>
<tr>
<td>server_hello_done</td>
<td>certificate</td>
</tr>
<tr>
<td>client_key_exchange</td>
<td>certificate_verify</td>
</tr>
</tbody>
</table>
Other key exchange methods (DH)

- **Anonymous Diffie-Hellman (DH)**
  - Public DH parameters \((Y_A, Y_B)\) are sent in `server_key_exchange` and `client_key_exchange` messages
  - The pre-master secret is the shared key computed with DH
    - No need to send it
    - No protection against man-in-the-middle attack, as DH parameters are not authenticated

- **Fixed Diffie-Hellman**
  - The DH public (key) parameters are fixed and signed by a CA
    - Resists to man-in-the-middle attack, but allows an attacker to use brute force on long-standing DH public-key parameters

- **Ephemeral Diffie-Hellman**
  - Ephemeral DH public-key parameters are exchanged
    - They can vary from session to session. More robust to brute force.
    - They are signed using the sender's private RSA key
      - Resists to man-in-the-middle thanks to this authentication of public-key parameters
      - Sender should have a secret RSA key to sign
      - So, client needs a certified RSA key too!

Master secret and keys

- The previous phases have generated a **pre-master secret**
  - Key chosen and sent encrypted to the server (with RSA)
  - Or, the DH secret key

- The **master secret** is generated (via pseudo random-number generator) from
  - The pre-master secret
  - The two nonces \((R_A, R_B)\) exchanged in the `client_hello` and `server_hello` messages
  - Makes it possible to use the same pre-master secret for several sessions (useful for Anonymous and Fixed DH, for example)

- **Six keys** are derived from this master secret:
  - Secret key used with MAC (for data sent by server)
  - Secret key used with MAC (for data sent by client)
  - Secret key and IV used for encryption (by server)
  - Secret key and IV used for encryption (by client)
**Server Gated Cryptography (SGC)**

- The US allows an exported client to use strong crypto when talking to some servers doing financial transactions
  - Those servers have an SGC certificate signed by Verisign (trusted by the Government, which turns out to be the real matter!)
  - This is wired in the implementation
  - Other trust authorities can be modified by the user
- Start with weak (exportable) cryptography, and then upgrade to strong cryptography if the server has an SGC certificate
  - The SGC certificate is discovered in phase 2 only
  - The client continues with a second handshake protected by the first master secret
  - Use Change_Cipher_Spec to switch to strong crypto
  - This does not require the server to run any special SGC code

**SSL Alert Protocol**

- This protocol is used to report errors
  - Examples
    - Unexpected message
    - Bad record MAC
    - Decompression failure
    - Handshake failure (i.e. security parameters negotiation failed)
    - Illegal parameters
- It is also used for other purposes
  - Examples
    - To notify closure of the TCP connection
    - To notify the absence of certificate (when requested)
    - To notify that a bad or unknown certificate was received
    - To notify that a certificate is revoked or has expired
Pros and Cons of Transport Layer Security

- **Pros**
  - Transport Layer Security is transparent to applications
  - Server is authenticated (if client’s browser correctly configured with trusted CAs to check server’s certificate)
  - Application layer headers are hidden
  - OK for direct client to server communication
  - More fine-grained than IPSec (see later) because it works at the transport connection level

- **Cons**
  - TCP/IP headers are in clear
  - Only applicable to secure TCP-based applications (not UDP)
  - Not enough to secure applications using intermediate servers and a chain of TCP connections (e.g. email)
  - Nonrepudiation not provided by SSL
  - Client authentication, if needed, must be implemented above SSL (e.g. username and password sent over the SSL connection) or client must have a certificate too

5: Securing TCP connections 39