CSE 484 / CSE M 584: Computer Security and Privacy

SSL/TLS

Fall 2016

Ada (Adam) Lerner
lerner@cs.washington.edu

Thanks to Franzi Roesner, Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...
We have all the pieces!

- Symmetric Encryption (privacy!)
- MACs (integrity!)
- Asymmetric Crypto (bootstrapping!)
- Certificate Authorities (authenticity!)
SSL/TLS

- Secure Sockets Layer and Transport Layer Security
  - Same protocol, new version (TLS is current)
- De facto standard for Internet security
  - “The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications”
- Deployed in every Web browser; also VoIP, payment systems, distributed systems, etc.
SSL/TLS

• TLS is typically used on top of a TCP connection

TLS

• Can be used over other transport protocols
TLS Basics

• TLS consists of **two** protocols
  – Familiar pattern for key exchange protocols

• Handshake protocol
  – Use *public-key cryptography* to establish a shared secret key between the client and the server

• Record protocol
  – Use the *secret symmetric key* established in the handshake protocol to protect communication between the client and the server
Basic Handshake Protocol

ClientHello

Client announces (in plaintext):
• Protocol version it is running
• Cryptographic algorithms it supports
• Fresh, random number
Basic Handshake Protocol

C, version\(_C\), suites\(_C\), N\(_C\)

ServerHello

Server responds (in plaintext) with:

- Highest protocol version supported by both the client and the server
- Strongest cryptographic suite selected from those offered by the client
- Fresh, random number
Basic Handshake Protocol

Server sends its **public-key certificate** containing either its RSA, or his Diffie-Hellman public key (depending on chosen crypto suite)
Basic Handshake Protocol

The client generates secret key material and sends it to the server encrypted with the server’s public key (if using RSA).
Basic Handshake Protocol

C, version\(_c\), suites\(_c\), N\(_c\)

\(\rightarrow\)

version\(_s\), suite\(_s\), N\(_s\), certificate, “ServerHelloDone”

\(\leftarrow\)

\{Secret\}_{PK_s} if using RSA

C and S share secret key material (secret\(_c\)) at this point

switch to keys derived from secret\(_c\), N\(_c\), N\(_s\)

\(\rightarrow\)

Finished

Record of all sent and received handshake messages

switch to keys derived from secret\(_c\), N\(_c\), N\(_s\)

\(\leftarrow\)

Finished
“Core” SSL 3.0 Handshake (Not TLS)

C, version\_c = 3.0, suites\_c, N\_c

\{Secret\_c\}_{PKs} if using RSA

C and S share secret key material (secret\_c) at this point

switch to keys derived from secret\_c, N\_c, N\_s

Finished

version\_s = 3.0, suite\_s, N\_s, certificate, “ServerHelloDone”

finished

switch to keys derived from secret\_c, N\_c, N\_s

Finished
Version Rollback Attack

C, version \(= 2.0\), suites, \(N_c\)

Server is fooled into thinking he is communicating with a client who supports only SSL 2.0

Version \(s=2.0\), suite, \(N_s\), certificate, “ServerHelloDone”

\(\{\text{Secret}_{c}\}_{PK_s}\)

C and S end up communicating using SSL 2.0 (weaker earlier version of the protocol that does not include “Finished” messages)
“Chosen-Protocol” Attacks

• Why do people release new versions of security protocols? Because the old version got broken!
• New version must be \textbf{backward-compatible}
  – Not everybody upgrades right away
• Attacker can fool someone into using the old, broken version and exploit known vulnerability
  – Similar: fool victim into using weak crypto algorithms
• Defense is hard: must authenticate version in early designs
• Many protocols have had “version rollback” attacks
  – SSL, SSH, GSM (cell phones)
Version Check in SSL 3.0

C, version\textsubscript{c}=3.0, suites\textsubscript{c}, N\textsubscript{c}

version\textsubscript{s}=3.0, suites\textsubscript{s}, N\textsubscript{s}, certificate for PK\textsubscript{s}, “ServerHelloDone”

“Embed” version number into secret

\{version\textsubscript{c}, secret\textsubscript{c}\}\textsubscript{PK\textsubscript{s}}

Check that received version is equal to the version in ClientHello

C and S share secret key material secret\textsubscript{c} at this point

switch to key derived from secret\textsubscript{c}, N\textsubscript{c}, N\textsubscript{s}

switch to key derived from secret\textsubscript{c}, N\textsubscript{c}, N\textsubscript{s}
The browser renders or executes arbitrary HTML, CSS, and Javascript send by hosts on the Internet.
Where Does the Attacker Live?

Browser

Malware attacker

Network attacker

request

website

Web attacker
All of These Should Be Safe

• Safe to visit an evil website

• Safe to visit two pages at the same time

• Safe delegation
Building Blocks of the Web (and Web Security)

• HTTP(S)
• Cookies
HTTP: HyperText Transfer Protocol

• Application layer protocol used by browsers and web servers

• Stateless request/response protocol
  – Each request is independent of previous requests
  – Statelessness has a significant impact on design and implementation of applications
HTTP Request

<table>
<thead>
<tr>
<th>Method</th>
<th>File</th>
<th>HTTP version</th>
<th>Headers</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/default.asp</td>
<td>HTTP/1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Accept: image/gif, image/x-bitmap, image/jpeg, <em>/</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Accept-Language: en</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Connection: Keep-Alive</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If-Modified-Since: Sunday, 17-Apr-96 04:32:58 GMT</td>
</tr>
</tbody>
</table>

Data – none for GET

Blank line
HTTP Response

HTTP version  Status code  Reason phrase

HTTP/1.0 200 OK
Date: Sun, 21 Apr 1996 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Content-Length: 2543

<HTML> Some data... blah, blah, blah </HTML>
HTTP Verbs

• HTTP declares a number of “verbs” that clients can use to request or provide information
  – **GET** asks for a resource
  – **POST** sends information
  – **HEAD** gets metadata (headers) for a resource

  – Also: PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH
HTTP Resources

• URL stands for **Uniform Resource Locator**
• Specifies the location of a resource on a network – what server is it on, where is it on that server?

• Resources could include HTML pages, images, data, etc.
HTTP Verbs

• HTTP declares a number of “verbs” that clients can use to request or provide information
  – **GET** asks for a resource
  – **POST** sends information
  – **HEAD** gets metadata (headers) for a resource
  – Also: PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH
HTTP Verbs

- HTTP declares a number of “verbs” that clients can use to request or provide information
  - **GET** asks for a resource *(Give me this image)*
  - **POST** sends information
  - **HEAD** gets metadata (headers) for a resource
  
  - Also: PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH
HTTP Verbs

• HTTP declares a number of “verbs” that clients can use to request or provide information
  – **GET** asks for a resource *(Give me this image)*
  – **POST** sends information *(I want to log in)*
  – **HEAD** gets metadata (headers) for a resource
  – Also: **PUT**, **DELETE**, **TRACE**, **OPTIONS**, **CONNECT**, **PATCH**
HTTP: HyperText Transfer Protocol

• Application layer protocol used by browsers and web servers

• **Stateless** request/response protocol
  – Each request is independent of previous requests
  – Statelessness has a significant impact on design and implementation of applications
Cookies – Statefulness for HTTP

A **cookie** is a file created by a website to store information in the browser.

HTTP is a stateless protocol; cookies add state.
Cookie Format

• Cookies are just KEY=VALUE pairs, e.g.,
  – `language=ENGLISH`
  – `userID=Alice`
  – `sessionID=8113d906-62e8-49e1-80e1-65805cb51cab`
  – `adID=9c740c60-8d88-4da6-bb83-041e95c1efac`
A **cookie** is a file created by a website to store information in the browser.

HTTP is a stateless protocol; cookies add state.

**Diagram:**
- **Browser** sends a POST request to `login.cgi` with username and pwd.
- **Server** sends a SET-cookie response with the following properties:
  - `NAME=VALUE`
  - `domain=(who can read)`
  - `expires=(when expires)`
  - `secure=(send only over HTTPS)`
- **Browser** requests `restricted.html` and includes the `SET-cookie` in the request.
- **Server** responds with the requested content and includes the `Cookie: NAME=VALUE` header.
- **If expires = NULL, this session only**
What Are Cookie Used For?

• Personalization
  – Website remembers visitor preferences
  – language=ENGLISH

• Authentication
  – The cookie “proves” client is logged in
  – sessionID=8113d906-62e8...

• Tracking
  – Follow the user from site to site;
  – adID=9c740c60-8d88...
Goals of Web Security

• Safely browse the Web
  – A malicious website cannot steal information from or modify legitimate sites or otherwise harm the user...
  – ... even if visited concurrently with a legitimate site -- in a separate browser window, tab, or even iframe on the same webpage

• Support secure Web applications
  – Applications delivered over the Web should have the same security properties we require for standalone applications
All of These Should Be Safe

• Safe to visit an evil website

• Safe to visit two pages at the same time

• Safe delegation
Two Sides of Web Security

- **Web browser**
  - Responsible for securely confining Web content presented by visited websites

- **Web applications**
  - Online merchants, banks, blogs, Google Apps …
  - Mix of server-side and client-side code
    - Server-side code written in PHP, Ruby, ASP, JSP… runs on the Web server
    - Client-side code written in JavaScript… runs in the Web browser
  - Many potential bugs: XSS, XSRF, SQL injection
Where Does the Attacker Live?

Attacker can make malicious requests to web servers – can even use HTML/JS to make those requests From users’ browsers!

Attacker gets to run Javascript and HTML code in the browser

Attacker may control 1 or more domains or websites
Web Attacker

- Controls a malicious website (attacker.com)
  - Can obtain an TLS certificate for attacker.com
- User visits attacker.com – why?
  - Phishing email, enticing content, search results, placed by an ad network, blind luck …
  - Or, attacker.com is embedded on another page
    - loading the friendly page loads content from attacker.com
Web Attacker

www.attacker.com
Javascript, or, Software Security for the Web!

```
<html>
  ...
  <p> The script on this page is totally trustworthy
  <script>
    doSomethingEvil()
  </script>
  ...
</html>
```

Browser receives content, displays HTML and executes scripts

www.attacker.com

A potentially malicious webpage gets to execute some code on user’s machine!
Browser Sandbox

• Goal: safely execute JavaScript provided by a website
  – No/limited access to OS/network/filesystem/browser data.

  – No buffer overflows, no way to execute arbitrary native code, process isolation between tabs

  – Attacker shouldn’t be able to access data from other tabs or browser windows

  – attacker.com shouldn’t be able to access data from bank.com, even if you’re logged in
A Strawperson Attack

www.attacker.com

www.bank.com (e.g., balance: $500)

www.attacker.com (the parent) cannot access HTML elements in the iframe (and vice versa).
Same-Origin Policy: DOM

Only code from same origin can access HTML elements on another site (or in an iframe).

- **www.example.com** (the parent) can access HTML elements in the iframe (and vice versa).
- **www.evil.com** (the parent) cannot access HTML elements in the iframe (and vice versa).
## Same-Origin Policy

Website origin = (scheme, domain, port)

<table>
<thead>
<tr>
<th>Compared URL</th>
<th>Outcome</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.example.com/dir/page.html">http://www.example.com/dir/page.html</a></td>
<td>Success</td>
<td>Same protocol and host</td>
</tr>
<tr>
<td><a href="http://www.example.com/dir2/other.html">http://www.example.com/dir2/other.html</a></td>
<td>Success</td>
<td>Same protocol and host</td>
</tr>
<tr>
<td><a href="http://www.example.com:81/dir/other.html">http://www.example.com:81/dir/other.html</a></td>
<td>Failure</td>
<td>Same protocol and host but different port</td>
</tr>
<tr>
<td><a href="https://www.example.com/dir/other.html">https://www.example.com/dir/other.html</a></td>
<td>Failure</td>
<td>Different protocol</td>
</tr>
<tr>
<td><a href="http://en.example.com/dir/other.html">http://en.example.com/dir/other.html</a></td>
<td>Failure</td>
<td>Different host</td>
</tr>
<tr>
<td><a href="http://example.com/dir/other.html">http://example.com/dir/other.html</a></td>
<td>Failure</td>
<td>Different host (exact match required)</td>
</tr>
<tr>
<td><a href="http://v2.www.example.com/dir/other.html">http://v2.www.example.com/dir/other.html</a></td>
<td>Failure</td>
<td>Different host (exact match required)</td>
</tr>
</tbody>
</table>

[Example thanks to Wikipedia.]