CSE 484 / CSE M 584: Computer Security and Privacy

Web Security: SSL/TLS

Spring 2015

Franziska (Franzi) Roesner
franzi@cs.washington.edu

Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...
SSL/TLS: More Details

- Secure Sockets Layer and Transport Layer Security protocols
  - Same protocol design, different crypto algorithms
- 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.
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 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

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

\[ C, \text{version}_C, \text{suites}_C, N_C \]
Basic Handshake Protocol

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

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

version\textsubscript{s}, suite\textsubscript{s}, N\textsubscript{s}, certificate, “ServerHelloDone”

ClientKeyExchange

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, versionₖ, suitesₖ, Nₛ

versionₛ, suiteₛ, Nₛ, certificate, “ServerHelloDone”

{Secretₖ}PKₛ if using RSA

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

switch to keys derived from secretₖ, Nₛ, Nₛ

switch to keys derived from secretₖ, Nₛ, Nₛ

Finished

Finished

Record of all sent and received handshake messages
“Core” SSL 3.0 Handshake

C, version\textsubscript{c}=3.0, suites\textsubscript{c}, N\textsubscript{c}  
\rightarrow version\textsubscript{s}=3.0, suite\textsubscript{s}, N\textsubscript{s},  
certificate,  
“ServerHelloDone”  

\{Secret\textsubscript{c}\}_PKs if using RSA  

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

\[\text{switch to keys derived from secret}_c, N_c, N_s\]  

Finished  

\[\text{switch to keys derived from secret}_c, N_c, N_s\]  

Finished
Version Rollback Attack

C, version \( = 2.0 \), suites \( \text{c} \), \( N_c \)

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

\( \text{Version}_s = 2.0 \), suite \( s \), \( N_s \), certificate, “ServerHelloDone”

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

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 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 had “version rollback” attacks
  - SSL, SSH, GSM (cell phones)
Version Check in SSL 3.0

C, version\(_c=3.0\), suites\(_c, N_c\)

```
version_s=3.0, suite_s, N_s, certificate for PK_s,
"ServerHelloDone"
```

"Embed" version number into secret

\{version\(_c, secret\(_c\)}_{PK_s}\n
C and S share secret key material secret\(_c\) at this point

switch to key derived from secret\(_c, N_c, N_s\)

Check that received version is equal to the version in ClientHello

switch to key derived from secret\(_c, N_c, N_s\)
Web Security: Basic Web Security Model

Spring 2015

Franziska (Franzi) Roesner
franzi@cs.washington.edu

Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials...
Browser and Network

Browser

OS

Hardware

Network

request

reply

website
HTTP: HyperText Transfer Protocol

• Used to request and return data
  – Methods: GET, POST, HEAD, ...

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

• Evolution
  – HTTP 1.0: simple
  – HTTP 1.1: more complex
## 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>GET /default.asp HTTP/1.0</td>
</tr>
</tbody>
</table>
|        |      |              | Accept: image/gif, image/x-bitmap, image/jpeg, */*
|        |      |              | Accept-Language: en |
|        |      |              | User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95) |
|        |      |              | Connection: Keep-Alive |
|        |      |              | If-Modified-Since: Sunday, 17-Apr-96 04:32:58 GMT |

Data – none for GET
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>
Websites Storing Info in Browser

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 the **Server**.
  - Request: `POST login.cgi username and pwd`.
  - **HTTP Header:** `Set-cookie:
    NAME=VALUE;
    domain = (who can read);
    expires = (when expires);
    secure = (send only over HTTPS)`.
  - **If expires = NULL, this session only**.

- **Browser** sends a **GET** request to the **Server**.
  - Request: `GET restricted.html Cookie: NAME=VALUE`.

---

**Notes:**

- **POST** vs. **GET** requests.
- Role of cookies in adding state.
- Security implications of cookies (domain, expires, secure).
What Are Cookies Used For?

• Authentication
  – The cookie proves to the website that the client previously authenticated correctly

• Personalization
  – Helps the website recognize the user from a previous visit

• Tracking
  – Follow the user from site to site; learn his/her browsing behavior, preferences, and so on
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
Security Vulnerabilities in 2011

Source: IBM X-Force
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?
Web Attacker

• Controls a malicious website (attacker.com)
  – Can even obtain an SSL/TLS certificate for his site
• User visits attacker.com – why?
  – Phishing email, enticing content, search results, placed by an ad network, blind luck …
• Attacker has no other access to user machine!
• Variation: “iframe attacker”
  – An iframe with malicious content included in an otherwise honest webpage
    • Syndicated advertising, mashups, etc.
<html>
  ...
  <p> The script on this page adds two numbers 
  <script>
    var num1, num2, sum
    num1 = prompt("Enter first number")
    num2 = prompt("Enter second number")
    sum = parseInt(num1) + parseInt(num2)
    alert("Sum = " + sum)
  </script>
  ...
  </html>

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

• Goal: safely execute JavaScript code provided by a website
  – No direct file access, limited access to OS, network, browser data, content that came from other websites

• Same origin policy
  – Can only access properties of documents and windows from the same domain, protocol, and port