Web security

HTTPS and the Lock Icon
Goals for this lecture

Brief overview of HTTPS:
  • How the SSL/TLS protocol works (very briefly)
  • How to use HTTPS

Integrating HTTPS into the browser
  • Lots of user interface problems to watch for
Threat Model: Network Attacker

Network Attacker:

- Controls network infrastructure: Routers, DNS
- Eavesdrops, injects, blocks, and modifies packets

Examples:

- Wireless network at Internet Café
- Internet access at hotels (untrusted ISP)
SSL/TLS overview

Public-key encryption:

- Bob generates \( (SK_{Bob}, PK_{Bob}) \)

- Alice: using \( PK_{Bob} \) encrypts messages and only Bob can decrypt
Certificates

How does Alice (browser) obtain $\text{PK}_{\text{Bob}}$?

- Browser Alice
  - $\text{PK}_{\text{CA}}$
  - Verify cert

- Server Bob
  - choose $(\text{SK}, \text{PK})$
  - $\text{PK}_{\text{CA}}$
  - $\text{PK}$ and proof “I am Bob”

- CA
  - check proof $\text{SK}_{\text{CA}}$

Bob’s key is PK

Bob’s key is PK

Bob uses Cert for an extended period (e.g. one year)
## Certificates: example

### Important fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial Number</td>
<td>5814744448373890497</td>
</tr>
<tr>
<td>Version</td>
<td>3</td>
</tr>
<tr>
<td>Signature Algorithm</td>
<td>SHA-1 with RSA Encryption (1.2.840.113549.1.1.5)</td>
</tr>
<tr>
<td>Parameters</td>
<td>none</td>
</tr>
<tr>
<td>Not Valid Before</td>
<td>Wednesday, July 31, 2013 4:59:24 AM Pacific Daylight Time</td>
</tr>
<tr>
<td>Not Valid After</td>
<td>Thursday, July 31, 2014 4:59:24 AM Pacific Daylight Time</td>
</tr>
<tr>
<td>Public Key Info</td>
<td></td>
</tr>
<tr>
<td>Algorithm</td>
<td>Elliptic Curve Public Key (1.2.840.10045.2.1)</td>
</tr>
<tr>
<td>Parameters</td>
<td>Elliptic Curve secp256r1 (1.2.840.10045.3.1.7)</td>
</tr>
<tr>
<td>Public Key</td>
<td>65 bytes: 04 71 6C DD E0 0A C9 76 ...</td>
</tr>
<tr>
<td>Key Size</td>
<td>256 bits</td>
</tr>
<tr>
<td>Key Usage</td>
<td>Encrypt, Verify, Derive</td>
</tr>
<tr>
<td>Signature</td>
<td>256 bytes: 8A 38 FE D6 F5 E7 F6 59 ...</td>
</tr>
</tbody>
</table>

---

[Certificate image]
Certificates on the web

Subject’s CommonName can be:

- An explicit name, e.g. `cs.stanford.edu`, or
- A wildcard cert, e.g. `*.stanford.edu` or `cs*.stanford.edu`

matching rules:

“*” must occur in leftmost component, does not match “.”

example: `*.a.com` matches `x.a.com` but not `y.x.a.com`

(as in RFC 2818: “HTTPS over TLS”)
### Certificate Authorities

Browsers accept certificates from a large number of CAs.

**Top level CAs** ≈ 60

**Intermediate CAs** ≈ 1200

<table>
<thead>
<tr>
<th>CA Name</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entrust.net Certification Authority</td>
<td>Jul 24, 2029 7:15:12 AM</td>
</tr>
<tr>
<td>Entrust.net Specification Authority</td>
<td>May 25, 2019 9:39:40 AM</td>
</tr>
<tr>
<td>ePKI Root Certification Authority</td>
<td>Dec 19, 2034 6:31:27 PM</td>
</tr>
<tr>
<td>Equifax Secure eBusiness CA-1</td>
<td>Jun 20, 2020 9:00:00 PM</td>
</tr>
<tr>
<td>Equifax Secure eBusiness CA-2</td>
<td>Jun 23, 2019 5:14:45 AM</td>
</tr>
<tr>
<td>Equifax Secure eBusiness CA-1</td>
<td>Jun 20, 2020 9:00:00 PM</td>
</tr>
<tr>
<td>Federal Common Policy CA</td>
<td>Dec 1, 2030 8:45:27 AM</td>
</tr>
<tr>
<td>FNMT Class 2 CA</td>
<td>Mar 18, 2019 8:26:19 AM</td>
</tr>
<tr>
<td>GeoTrust Global CA</td>
<td>May 20, 2022 9:00:00 PM</td>
</tr>
<tr>
<td>GeoTrust Certification Authority</td>
<td>Jul 16, 2036 4:59:59 PM</td>
</tr>
<tr>
<td>Global Chambersign Root</td>
<td>Sep 30, 2037 9:14:18 AM</td>
</tr>
</tbody>
</table>
Brief overview of SSL/TLS

Browser

client-hello

server-hello + server-cert (PK)

key exchange (several options): EC-DHE

server-key-exchange

client-key-exchange

Finished

HTTP data encrypted with KDF(k)

Most common: server authentication only
Integrating SSL/TLS with HTTP: HTTPS

Two complications

Web proxies
solution: browser sends `CONNECT domain-name` before client-hello

Virtual hosting:
two sites hosted at same IP address.
solution in TLS 1.1: SNI (June 2003)
`client_hello_extension: server_name=cnn.com`
implemented since FF2 and IE7 (vista)
Why is HTTPS not used for all web traffic?

• Crypto slows down web servers  (but not by much if done right)

• Some ad-networks do not support HTTPS  (2015 stats: 20%)
  – Reduced revenue for publishers

• Incompatible with virtual hosting  (older browsers)
  March 2015:  IE6 ≈ 1%  (ie6countdown.com)

Aug 2014:  Google boosts ranking of sites supporting HTTPS
HTTPS in the Browser
The lock icon: SSL indicator

Intended goal:

• Provide user with identity of page origin
• Indicate to user that page contents were not viewed or modified by a network attacker

In reality: many problems (next few slides)
When is the (basic) lock icon displayed

All elements on the page fetched using HTTPS

For all elements:

• HTTPS cert issued by a CA trusted by browser
• HTTPS cert is valid (e.g. not expired)
• Domain in URL matches: CommonName or SubjectAlternativeName in cert
The lock UI: Extended Validation Certs

Harder to obtain than regular certs

- requires human at CA to approve cert request
- no wildcard certs (e.g. *.stanford.edu)

Helps block “semantic attacks”: www.bankofthevvest.com

note: HTTPS-EV and HTTPS are in the same origin
A general UI attack: picture-in-picture

Trained users are more likely to fall victim to this  [JSTB’07]
HTTPS and login pages: incorrect usage

Users often land on login page over HTTP:

• Type HTTP URL into address bar

• Google links to HTTP page

View source:

```html
<form method="post" action="https://onlineservices.wachovia.com/...">
```

( old site)
HTTPS and login pages: guidelines

General guideline:

Response to http://login.site.com should be Location: https://login.site.com (redirect)
Problems with HTTPS and the Lock Icon
Problems with HTTPS and the Lock Icon

1. Upgrade from HTTP to HTTPS

2. Forged certs

3. Mixed content: HTTP and HTTPS on the same page

4. Does HTTPS hide web traffic?
   - Problems: traffic analysis, compression attacks
1. HTTP $\Rightarrow$ HTTPS upgrade

Common use pattern:
- browse site over HTTP; move to HTTPS for checkout
- connect to bank over HTTP; move to HTTPS for login

SSL Strip attack: prevent the upgrade [Moxie’08]
Tricks and Details

Tricks: drop-in a clever fav icon (older browsers)

⇒ fav icon no longer presented in address bar

More tricks: inject “Set-cookie” headers to delete existing session cookies in browser. Force login.

Number of users who detected HTTP downgrade: 0
Defense: Strict Transport Security (HSTS)

Header tells browser to always connect over HTTPS

Subsequent visits must be over HTTPS (self signed certs result in an error)

- Browser refuses to connect over HTTP or if self-signed cert
- Requires that entire site be served over HTTPS

HSTS flag deleted when user “clears private data”: security vs. privacy
CSP: upgrade-insecure-requests

The problem: many pages use <img src="http://site.com/img">  
• Makes it difficult to migrate a section of a site to HTTPS

Solution: gradual transition using CSP

Content-Security-Policy: upgrade-insecure-requests

Always use protocol relative URLs
2. Certificates: wrong issuance

2011: **Comodo** and **DigiNotar** CAs hacked, issue certs for Gmail, Yahoo! Mail, ...

2013: **TurkTrust** issued cert. for gmail.com (discovered by pinning)

2014: **Indian NIC** (intermediate CA trusted by the root CA **IndiaCCA**) issue certs for Google and Yahoo! domains

Result:
1. India CCA revoked NIC’s intermediate certificate
2. Chrome restricts India CCA root to only seven Indian domains

2015: **MCS** (intermediate CA cert issued by **CNNIC**) issues certs for Google domains

Result: current CNNIC root no longer recognized by Chrome

⇒ enables eavesdropping w/o a warning on user’s session
Man in the middle attack using rogue cert

GET https://bank.com

ClientHello

ServerCert (rogue)

BankCert

BadguyCert

ClientHello

ServerCert (Bank)

(cert for Bank by a valid CA)

SSL key exchange

k₁

HTTP data enc with k₁

k₁

SSL key exchange

k₂

HTTP data enc with k₂

k₂

Attacker proxies data between user and bank.
Sees all traffic and can modify data at will.
What to do? (many good ideas)

1. Dynamic HTTP public-key pinning (RFC 7469)
   - Let a site declare CAs that can sign its cert (similar to HSTS)
   - on subsequent HTTPS, browser rejects certs issued by other CAs
   - TOFU: Trust on First Use

2. Certificate Transparency: [LL’12]
   - idea: CA’s must advertise a log of all certs. they issued
   - Browser will only use a cert if it is published on log server
     • Efficient implementation using Merkle hash trees
     • Companies can scan logs to look for invalid issuance
HPKP example  (HTTP header from server)

Public-Key-Pins: max-age=2592000;
   pin-sha256="E9CZ9INDbd+2eRQozYqqbQ2yXLVKB9+xcprMF+44U1g=";
   pin-sha256="LPJNuI+wow4m6DsqxbninhsWHIwp0JecwQzYpOLmCQ=";
   report-uri="https://example.net/pkp-report"

Examine browser’s pinning DB:  chrome://net-internals/#hsts
Page loads over HTTPS, but contains content over HTTP
(e.g. <script src="http://.../script.js">)

⇒ Active network attacker can hijack session by modifying script en-route to browser

IE7:

Old Chrome:

Chrome policy: blocked: CSS, script, frame; allowed: images, XHR
4. Peeking through SSL: traffic analysis

- Network traffic reveals length of HTTPS packets
  - TLS supports up to 256 bytes of padding

- AJAX-rich pages have lots and lots of interactions with the server

- These interactions expose specific internal state of the page

BAM!

Chen, Wang, Wang, Zhang, 2010
Peeking through SSL: an example [CWWZ’10]

Vulnerabilities in an online tax application

No easy fix. Can also be used to ID Tor traffic
Peeking through SSL: compression [DR’12]

HTTPS: supports compressing data before encryption (16KB records)

Attacker: wants to recover Gmail session cookie (say)

• Places Javascript on some site that issues request:

```
GET gmail.com/__AAAAAAAAAAAAA....AAAAAA
Cookie: session=__A6Bh63g53ig4
Host: gmail.com
```

1\textsuperscript{st} byte of cookie is “A” \Rightarrow record will compress more than when not

• Script tries all possibilities to expose 1\textsuperscript{st} byte. Moves to 2\textsuperscript{nd} bytes …

What to do: do not use compression with HTTPS
THE END