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)
TLS overview: (1) DH key exchange

Anonymous key exchange secure against eavesdropping:

The Diffie-Hellman protocol in a group $G = \{1, g, g^2, g^3, \ldots, g^{q-1}\}$

Browser Alice

- $a \leftarrow \{1, \ldots, q\}$
- $PMS = B^a = g^{ab} = (g^b)^a = B^a = (g^a)^b = A^b$

Server Bob

- $b \leftarrow \{1, \ldots, q\}$
- $A = g^a \in G$
- $B = g^b \in G$
- $PMS = A^b$
(2) Digital signatures

Goal: bind document to author
  • Problem: attacker can copy Alice’s sig from one doc to another

Main idea: make signature depend on contents of document

**Def:** a signature scheme is a tuple of three algorithms:

- **Gen()** \(\rightarrow\) (pk, sk)
  - sign msg using sk

- **Sign(sk, msg)** \(\rightarrow\) *sig*

- **Verify(pk, msg, sig)** \(\rightarrow\) ‘accept’ or ‘reject’
(2) Digital signatures

**Security** (informal): adversary who sees signatures on many messages of its choice, cannot forge a signature on a new message.

**Def:** a signature scheme is a tuple of three algorithms:

- **Gen()** $\rightarrow$ (pk, sk)
- **Sign(sk, msg)** $\rightarrow$ *sig*
- **Verify(pk, msg, sig)** $\rightarrow$ ‘accept’ or ‘reject’
(3) Certificates

How does Alice (browser) obtain $pk_{Bob}$?

Browser Alice

$pk_{CA}$

Verify cert

Server Bob

choose $(sk,pk)$

$pk_{CA}$

CA

check proof

$sk_{CA}$

Bob uses Cert for an extended period (e.g. one year)

Bob’s key is $pk$

Bob’s key is $pk$

Bob uses Cert for an extended period (e.g. one year)
Sample certificate:

Organization: Bank of America Corporation
Business Category: Private Organization
Organizational Unit: eComm Network Infrastructure
Serial Number: 2927442
Common Name: www.bankofamerica.com

Public Key Info:
Algorithm: RSA (1.2.840.113549.1.1.1)
Parameters: None
Public Key: 256 bytes: BE E5 23 1D 17 9A 68 05 ...
Exponent: 65537
Key Size: 2,048 bits
Key Usage: Encrypt, Verify, Wrap, Derive

Signature:
(by CA) 256 bytes: 39 D0 09 7E 99 C6 B3 01 ...
Certificates on the web

Subject’s CommonName can be:

- An explicit name, e.g. \texttt{cs.stanford.edu}, or
- A wildcard cert, e.g. \texttt{*\.stanford.edu} or \texttt{cs*.stanford.edu}

matching rules:

\texttt{“*”} must occur in leftmost component, does not match \texttt{“.”}

example: \texttt{*\.a.com} matches \texttt{x\.a\.com} but not \texttt{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
(4) **TLS 1.3 session setup** (simplified)

**Client**

- **ClientHello**: nonce\(_C\), KeyShare

**Server**

- **ServerHello**: nonce\(_S\), KeyShare, \(\text{Enc}[\text{cert}_S, \ldots]\)
- **CertVerify**: \(\text{Enc}[	ext{Sig}_S(\text{data})]\), Finished
- **Finished**

**Secret key**

- \(\text{cert}_S\)

**Session keys**

- \(\text{session-keys} \leftarrow \text{HKDF}(\text{DHkey}, \text{nonce}_C, \text{nonce}_S)\)

**Encrypted ApplicationData**

- Most common: server authentication only

---

Dan Boneh
(3) TLS 1.3 session setup: optimization (and caution)

ClientHello: nonce_c, KeyShare, Enc[0-RTT data]

ServerHello: nonce_s, KeyShare, [Enc[cert_s, ...]]

CertVerify: Enc[Sig_s(data)]

Finished

Client

Server

secret key
cert_s

Data encrypted using a pre-shared key

**Caution**: 0-RTT data is vulnerable to replay

⇒ data should have no side effects

(i.e. GET but not POST)

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

HTTP/2
(2015)

TCP
IP
TLS
HTTP

TCP handshake ⇒ TLS handshake

HTTP/3
(2022)

IP
QUIC
TLS

integrated handshake
(encrypting by default)
Integrating TLS with HTTP: HTTPS

A complication:

**Virtual hosting**: many sites hosted at same IP address

Solution since TLS 1.1: SNI (2003)

client_hello_extension: `server_name=cnn.com`

... but SNI defeats privacy benefit of encrypted cert in TLS 1.3.

Solution: **enc. client hello (ECH)** [encrypted with pk in server DNS]
HTTPS for all web traffic?

Old excuse:

- Crypto slows down web servers

  ⇒ no longer true (thanks to AES-NI)

Since July 2018:  Chrome marks HTTP sites as insecure
HTTPS in the Browser
The TLS 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
When is the TLS 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
Positive security indicators are dangerous

The lock icon is a **positive security indicator**. Problem: picture-in-picture attacks.

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

Suppose user lands on HTTP login page.

- say, by typing HTTP URL into address bar

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)

Should be the response to every HTTP request...
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 ⇒ HTTPS upgrade

Suppose user does:
  • connect to bank site over HTTP; bank redirects to HTTPS

SSL_strip attack: prevent the upgrade [Moxie’08]

```
<a href=https://...> ⟶ <a href=http://...>
Location: https://... ⟶ Location: http://... (redirect)
<form action=https://... > ⟶ <form action=http://...>
```
Tricks and Details

UI design flaw in old browsers: location of fav icon

⇒ fav icon no longer presented in address bar

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

Strict-Transport-Security: max-age=63072000; includeSubDomains
(ignored if not over HTTPS)

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 site presents an invalid cert
- Requires that entire site be served over valid HTTPS

HSTS flag deleted when user “clears private data” : security vs. privacy
Preloaded HSTS list

https://hstspreload.org/

Strict-Transport-Security: max-age=63072000; includeSubDomains; preload

Preload list hard-coded in Chrome source code. Examples:
Google, Paypal, Twitter, Simple, Linode, Stripe, Lastpass, …
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

```html
<img src="http://site.com/img">
<img src="http://othersite.com/img">
<a href="http://site.com/img">
<a href="http://othersite.com/img">
```

```html
<img src="https://site.com/img">
<img src="https://othersite.com/img">
<a href="https://site.com/img">
<a href="https://othersite.com/img">
```
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

2016: **WoSign** (Chinese CA) issues cert for GitHub domain (among other issues)

Result: WoSign certs no longer trusted by Chrome and Firefox

⇒ 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)

attaacker

ClientHello

ServerCert (Bank)

bank

(cert for Bank by a valid CA)

 TLS key exchange

$k_1$ $k_1$

HTTP data enc with $k_1$

 TLS key exchange

$k_2$ $k_2$

HTTP data enc with $k_2$

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

1. Public-key pinning (static pins)
   - Hardcode list of allowed CAs for certain sites (Gmail, Facebook, ...)
   - Browser rejects certs issued by a CA not on list

2. Certificate Transparency (CT): [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 (two) log servers
     • Server attaches to certificate a signed statement from log (SCT)
     • Companies can scan logs to look for invalid issuance
CT requirements

April 30, 2018: CT required by chrome

• Required for all certificates with a path to a trusted root CA
  (not required for an installed root CA)

• Otherwise: HTTPS errors

Cert for crypto.stanford.edu published on five logs:
  cloudflare_nimbus2018
  google_argon2018, google_aviator
  google_pilot, google_rocketeer
3. Mixed Content: HTTP and HTTPS

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:

Mostly ignored by users ...
https://badssl.com  
(Chrome 124, 2024)

Mixed script:  
<script src="http://mixed-script.badssl.com/nonsecure.js"></script>

mixed-script.badssl.com  
(no visible warning)

script is not loaded!  developer tools show an error.

Mixed form:  
<form action="http://http.badssl.com/resources/submit.html">

mixed-form.badssl.com

Warning if user tries to submit data

The information you’re about to submit is not secure
Because this form is being submitted using a connection that’s not secure, your information will be visible to others.
4. Peeking through TLS: traffic analysis

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

• Some sites interact frequently with the web server
  – These interactions expose specific internal state of the page

Chen, Wang, Wang, Zhang, 2010  (tax web site)
IOactive, 2012  (Google maps)
Peeking through TLS: an example [CWWZ’10]

Vulnerabilities in an online tax application

No easy fix. Can also be used to ID Tor traffic
THE END