Cookies and Session Management

Dan Boneh
Same origin policy: “high level”

Review: Same Origin Policy (SOP) for DOM:
- Origin A can access origin B’s DOM if match on $(scheme, domain, port)$

Today: Same Original Policy (SOP) for cookies:
- Generally speaking, based on: $(\text{[scheme]}, \text{domain}, \text{path})$

scheme://domain:port/path?params
Setting/deleting cookies by server

- Delete cookie by setting “expires” to date in past
- Default scope is domain and path of setting URL

HTTP Header:
Set-cookie: NAME=VALUE ;
domain = (when to send) ;
path = (when to send) ;
secure = (only send over SSL) ;
extpires = (when expires) ;
HttpOnly

if expires=NULL: this session only
Scope setting rules (write SOP)

**domain:** any domain-suffix of URL-hostname, except TLD

**example:** host = “login.site.com”

---

<table>
<thead>
<tr>
<th>allowed domains</th>
<th>disallowed domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>login.site.com</td>
<td>user.site.com</td>
</tr>
<tr>
<td>.site.com</td>
<td>othersite.com</td>
</tr>
<tr>
<td>.com</td>
<td>.com</td>
</tr>
</tbody>
</table>

⇒ login.site.com can set cookies for all of .site.com but not for another site or TLD

Problematic for sites like .stanford.edu

**path:** can be set to anything
Cookies are identified by (name, domain, path)

- **cookie 1**
  - name: `userid`
  - value: `test`
  - domain: `login.site.com`
  - path: `/secure`

- **cookie 2**
  - name: `userid`
  - value: `test123`
  - domain: `.site.com`
  - path: `/secure`

Both cookies stored in browser’s cookie jar; both are in scope of `login.site.com`
Reading cookies on server (read SOP)

Browser sends all cookies in URL scope:

- cookie-domain is domain-suffix of URL-domain, and
- cookie-path is prefix of URL-path, and
- [protocol=HTTPS if cookie is “secure”]

Goal: server only sees cookies in its scope
Examples

both set by login.site.com

<table>
<thead>
<tr>
<th>cookie 1</th>
<th>cookie 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>name = \textbf{userid}</td>
<td>name = \textbf{userid}</td>
</tr>
<tr>
<td>value = \texttt{u1}</td>
<td>value = \texttt{u2}</td>
</tr>
<tr>
<td>domain = login.site.com</td>
<td>domain = .site.com</td>
</tr>
<tr>
<td>path = /</td>
<td>path = /</td>
</tr>
<tr>
<td>secure</td>
<td>non-secure</td>
</tr>
</tbody>
</table>

http://checkout.site.com/          cookie: userid=u2
http://login.site.com/             cookie: userid=u2
https://login.site.com/            \textbf{cookie: userid=u1; userid=u2}
                                    (arbitrary order)
Client side read/write:  document.cookie

- Setting a cookie in Javascript:
  document.cookie = “name=value; expires=...; ”

- Reading a cookie:  alert(document.cookie)
  prints string containing all cookies available for document  (based on [protocol], domain, path)

- Deleting a cookie:
  document.cookie = “name=; expires= Thu, 01-Jan-70”

document.cookie often used to customize page in Javascript
Javascript URL

Javascript: alert(document.cookie)

Displays all cookies for current document
### Viewing/deleting cookies in Browser UI

A screenshot of a browser's cookie management interface, showing a list of cookies for `google.com`. The table includes columns for `Site`, `Cookie Name`, `Content`, `Domain`, `Path`, `Send For`, and `Expires`.

- **Site**: google.com, google.com, google.com, google.com
- **Cookie Name**: _utma, _utmc, _utmb, _utua, _utmt
- **Content**: Various values, including timestamps
- **Domain**: .google.com
- **Path**: /adsense/
- **Send For**: Any type of connection
- **Expires**: Varies, with one example: Sunday, January 17, 2038 4:00:00 PM

Buttons for removing a single cookie or all cookies are present at the bottom of the window.
Cookie protocol problems

Server is blind:
- Does not see cookie attributes (e.g. secure)
- Does not see which domain set the cookie

Server only sees:  

Cookie:  NAME=VALUE
Example 1: login server problems

• Alice logs in at \texttt{login.site.com}

  \texttt{login.site.com} sets session-id cookie for \texttt{.site.com}

• Alice visits \texttt{evil.site.com}

  overwrites \texttt{.site.com} session-id cookie
  with session-id of user “badguy”

• Alice visits \texttt{cs241.site.com} to submit homework.

  \texttt{cs241.site.com} thinks it is talking to “badguy”

Problem: \texttt{cs241} expects session-id from \texttt{login.site.com};
  cannot tell that session-id cookie was overwritten
Example 2: “secure” cookies are not secure

Alice logs in at \texttt{https://www.google.com/accounts}

Alice visits \texttt{http://www.google.com} (cleartext)

- Network attacker can inject into response
  \texttt{Set-Cookie: LSID=badguy; secure}
  and overwrite secure cookie

Problem: network attacker can re-write HTTPS cookies!
⇒ HTTPS cookie value cannot be trusted
Interaction with the DOM SOP

Cookie SOP: path separation
\texttt{x.com/A} does not see cookies of \texttt{x.com/B}

Not a security measure:
DOM SOP: \texttt{x.com/A} has access to DOM of \texttt{x.com/B}

\begin{verbatim}
<iframe src="x.com/B"></iframe>
alert(frames[0].document.cookie);
\end{verbatim}

Path separation is done for efficiency not security:
\texttt{x.com/A} is only sent the cookies it needs
Cookies have no integrity!!
Storing security data on browser?

- User can change and delete cookie values !!
  - Edit cookie file (FF3: cookies.sqlite)
  - Modify Cookie header (FF: TamperData extension)

- Silly example: shopping cart software
  
  Set-cookie: shopping-cart-total = 150 ($)

- User edits cookie file (cookie poisoning):
  
  Cookie: shopping-cart-total = 15 ($)

Similar to problem with hidden fields

<INPUT TYPE="hidden" NAME=price VALUE="150">
Not so silly ...  (as of 2/2000)

- D3.COM Pty Ltd: ShopFactory 5.8
- @Retail Corporation: @Retail
- Adgrafix: Check It Out
- Baron Consulting Group: WebSite Tool
- ComCity Corporation: SalesCart
- Crested Butte Software: EasyCart
- Dansie.net: Dansie Shopping Cart
- Intelligent Vending Systems: Intellivend
- Make-a-Store: Make-a-Store OrderPage
- McMurtrey/Whitaker & Associates: Cart32 3.0
- pknutsen@nethut.no: CartMan 1.04
- Rich Media Technologies: JustAddCommerce 5.0
- SmartCart: SmartCart
- Web Express: Shoptron 1.2

Source: http://xforce.iss.net/xforce/xfdb/4621
Solution: cryptographic checksums

Goal: data integrity
Requires secret key \( k \) unknown to browser

- **Generate tag:** \( T \leftarrow F(k, \text{value}) \)
- **Verify tag:** \( T = F(k, \text{value}) ? \)

“value” should also contain data to prevent cookie replay and swap
Example: .NET 2.0

  - Secret web server key intended for cookie protection
  - Stored on all web servers in site

Creating an encrypted cookie with integrity:

```csharp
HttpCookie cookie = new HttpCookie(name, val);
HttpCookie encodedCookie = HttpSecureCookie.Encode(cookie);
```

Decrypting and validating an encrypted cookie:

```csharp
HttpSecureCookie.Decode(cookie);
```
Session management
Sessions

- A sequence of requests and responses from one browser to one (or more) sites
  - Session can be long (Gmail - two weeks) or short
  - without session mgmt: users would have to constantly re-authenticate

Session mgmt:
- Authorize user once;
- All subsequent requests are tied to user
Pre-history: HTTP auth

HTTP request: GET /index.html

HTTP response contains:

WWW-Authenticate: Basic realm="Password Required"

Browsers send hashed password on all subsequent HTTP requests:

Authorization: Basic ZGFddfibzsdfgkjheczI1NXRlheHQ=
HTTP auth problems

- Hardly used in commercial sites
  - User cannot log out other than by closing browser
    - What if user has multiple accounts?
    - What if multiple users on same computer?
  - Site cannot customize password dialog
  - Confusing dialog to users
  - Easily spoofed
  - Defeated using a TRACE HTTP request (on old browsers)
Session tokens

Browser

GET /index.html
set anonymous session token

GET /books.html
anonymous session token

POST /do-login
Username & password
elevate to a logged-in session token

POST /checkout
logged-in session token

Web Site

check credentials (later)
Validate token
Storing session tokens:  
Lots of options  
(but none are perfect)

• Browser cookie:

    Set-Cookie: SessionToken=fduhye63sfdb

• Embedd in all URL links:

    https://site.com/checkout ? SessionToken=kh7y3b

• In a hidden form field:

    <input type="hidden" name="sessionid" value="kh7y3b">

• Window.name DOM property
Storing session tokens: problems

• Browser cookie:
  browser sends cookie with every request, even when it should not (CSRF)

• Embed in all URL links:
  token leaks via HTTP Referer header

• In a hidden form field: short sessions only

Best answer: a combination of all of the above.
why? next lecture.
The HTTP referer header

GET /wiki/John_Ousterhout HTTP/1.1
Host: en.wikipedia.org
Keep-Alive: 300
Connection: keep-alive

Referer: http://www.google.com/search?q=john+ousterhout&ie=utf-8&oe=utf-8

Referer leaks URL session token to 3rd parties
SESSION HIJACKING

Attacker waits for user to login; then attacker obtains user’s Session Token and “hijacks” session.
1. Predictable tokens

Example: counter (Verizon Wireless)
⇒ user logs in, gets counter value, can view sessions of other users

Example: weak MAC (WSJ)
- token = \{userid, MAC_k(userid)\}
- Weak MAC exposes k from few cookies.

Session tokens must be unpredictable to attacker:
Use underlying framework.

Rails: token = MD5( current time, random nonce )
2. Cookie theft

Example 1: login over SSL, but subsequent HTTP
- What happens at wireless Café?
- Other reasons why session token sent in the clear:
  - HTTPS/HTTP mixed content pages at site
  - Man-in-the-middle attacks on SSL

Example 2: Cross Site Scripting (XSS) exploits

Amplified by poor logout procedures:
- Logout must invalidate token on server
Session fixation attacks

Suppose attacker can set the user’s session token:
- For URL tokens, trick user into clicking on URL
- For cookie tokens, set using XSS exploits

**Attack:** (say, using URL tokens)
1. Attacker gets anonymous session token for site.com
2. Sends URL to user with attacker’s session token
3. User clicks on URL and logs into site.com
   - this elevates attacker’s token to logged-in token
4. Attacker uses elevated token to hijack user’s session.
Session fixation: lesson

- When elevating user from anonymous to logged-in, always issue a new session token.
  - Once user logs in, token changes to value unknown to attacker.
  - ⇒ Attacker’s token is not elevated.
Generating session tokens

Goal: prevent hijacking and avoid fixation
Option 1: minimal client-side state

- SessionToken = [random unpredictable string]
  (no data embedded in token)

  - Server stores all data associated to SessionToken:
    userid, login-status, login-time, etc.

- Can result in server overhead:
  - When multiple web servers at site, lots of database lookups to retrieve user state.
Option 2: lots of client-side state

- **SessionToken:**
  
  $\text{SID} = [\text{userID, exp. time, data}]$
  
  where $\text{data} = (\text{capabilities, user data, ...})$
  
  $\text{SessionToken} = \text{Enc-then-MAC (k, SID)}$

  (as in CS255)

  $k$: key known to all web servers in site.

- Server must still maintain some user state:
  - e.g. logout status (should be checked on every request)

- Note that nothing binds SID to client’s machine
Binding SessionToken to client’s computer; mitigating cookie theft

**approach:** embed machine specific data in SID

**Client IP Address:**
- Will make it harder to use token at another machine
- But honest client may change IP addr during session
  - client will be logged out for no reason.

**Client user agent:**
- A weak defense against theft, but doesn’t hurt.

**SSL session key:**
- Same problem as IP address  
  (and even worse)
User Authentication

... next lecture
Attempts at defeating key-loggers

Bank of Adelaide

Swivel PinSafe