Web Security:
Session management

CS 161: Computer Security

Prof. Raluca Ada Popa

April 10, 2020

Some content adapted from materials by David Wagner or Dan Boneh
Cookies

A way of maintaining state in the browser

Browser

GET ...

Server

http response contains

Browser maintains cookie jar with all cookies it receives
Setting/deleting cookies by server

- The first time a browser connects to a particular web server, it has no cookies for that web server.
- When the web server responds, it includes a `Set-Cookie:` header that defines a cookie.
- Each cookie is just a name-value pair (with some extra metadata).
View a cookie

In a web console (firefox, tool->web developer->web console), type

    document.cookie

to see the cookie for that site

Each name=value is one cookie. document.cookie lists all cookies in scope for document
When the browser connects to the same server later, it automatically attaches the cookies in scope: header containing the name and value, which the server can use to connect related requests.

Domain and path inform the browser about which sites to send this cookie to
HTTP Header:

```
Set-cookie: NAME=VALUE ;
  domain = (when to send) ;
  path = (when to send) ;
  secure = (only send over HTTPS);
```

- **Secure**: sent over https only
  - https provides secure communication using TLS (privacy and integrity)
Cookie scope

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

• Expires is expiration date
  • Delete cookie by setting “expires” to date in past

• HttpOnly: cookie cannot be accessed by Javascript, but only sent by browser (defense in depth, but does not prevent XSS)
Cookie policy

The cookie policy has two parts:
1. What scopes a URL-host name web server is allowed to set on a cookie
2. When the browser sends a cookie to a URL
Cookie scope

• Scope of cookie might not be the same as the URL-host name of the web server setting it
What scope a server may set for a cookie

The browser checks if the web server may set the cookie, and if not, it will not accept the cookie.

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

**example:** host = “login.site.com”

**allowed domains**
- login.site.com
- .site.com

**disallowed domains**
- user.site.com
- othersite.com
- .com

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

Problematic for sites like *.berkeley.edu*

**path:** can be set to anything
Examples

Web server at foo.example.com wants to set cookie with domain:

<table>
<thead>
<tr>
<th>domain</th>
<th>Whether it will be set, and if so, where it will be sent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>(value omitted)</td>
<td>foo.example.com (exact)</td>
</tr>
<tr>
<td>bar.foo.example.com</td>
<td>Cookie not set: domain more specific than origin</td>
</tr>
<tr>
<td>foo.example.com</td>
<td>*.foo.example.com</td>
</tr>
<tr>
<td>baz.example.com</td>
<td>Cookie not set: domain mismatch</td>
</tr>
<tr>
<td>example.com</td>
<td>*.example.com</td>
</tr>
<tr>
<td>ample.com</td>
<td>Cookie not set: domain mismatch</td>
</tr>
<tr>
<td>.com</td>
<td>Cookie not set: domain too broad, security risk</td>
</tr>
</tbody>
</table>

When browser sends cookie

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
When browser sends cookie

GET //URL-domain/URL-path
Cookie: NAME = VALUE

A cookie with
domain = example.com, and
path = /some/path/
will be included on a request to
http://foo.example.com/some/path/subdirectory/hello.txt
Examples: Which cookie will be sent?

**cookie 1**
name = **userid**
value = **u1**
domain = **login.site.com**
path = /
non-secure

**cookie 2**
name = **userid**
value = **u2**
domain = **.site.com**
path = /
non-secure

http://checkout.site.com/      cookie: **userid=u2**
http://login.site.com/        cookie: **userid=u1, userid=u2**
http://othersite.com/        cookie: **none**
### Examples

Web server at foo.example.com wants to set cookie with domain:

<table>
<thead>
<tr>
<th>domain</th>
<th>Whether it will be set, and if so, where it will be sent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>(value omitted)</td>
<td>foo.example.com (exact)</td>
</tr>
<tr>
<td>bar.foo.example.com</td>
<td>Cookie not set: domain more specific than origin</td>
</tr>
<tr>
<td>foo.example.com</td>
<td>?</td>
</tr>
<tr>
<td>baz.example.com</td>
<td>Cookie not set: domain mismatch</td>
</tr>
<tr>
<td>example.com</td>
<td>?</td>
</tr>
<tr>
<td>ample.com</td>
<td>Cookie not set: domain mismatch</td>
</tr>
<tr>
<td>.com</td>
<td>Cookie not set: domain too broad, security risk</td>
</tr>
</tbody>
</table>

Web server at foo.example.com wants to set cookie with domain:

<table>
<thead>
<tr>
<th>domain</th>
<th>Whether it will be set, and if so, where it will be sent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>(value omitted)</td>
<td>foo.example.com (exact)</td>
</tr>
<tr>
<td>bar.foo.example.com</td>
<td>Cookie not set: domain more specific than origin</td>
</tr>
<tr>
<td>foo.example.com</td>
<td>*.foo.example.com</td>
</tr>
<tr>
<td>baz.example.com</td>
<td>Cookie not set: domain mismatch</td>
</tr>
<tr>
<td>example.com</td>
<td>*.example.com</td>
</tr>
<tr>
<td>ample.com</td>
<td>Cookie not set: domain mismatch</td>
</tr>
<tr>
<td>.com</td>
<td>Cookie not set: domain too broad, security risk</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>cookie 1</th>
<th>cookie 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>name = userid</td>
<td>name = userid</td>
</tr>
<tr>
<td>value = u1</td>
<td>value = 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/          cookie: userid=u1; userid=u2
                                 (arbitrary order)
Client side read/write:  \texttt{document\textunderscore cookie}

- Setting a cookie in Javascript:
  \begin{verbatim}
  document.cookie = "name=value; expires=…; ";
  \end{verbatim}

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

- Deleting a cookie:
  \begin{verbatim}
  document.cookie = "name=; expires= Thu, 01-Jan-00"
  \end{verbatim}

\texttt{document\textunderscore cookie} often used to customize page in Javascript
Viewing/deleting cookies in Browser UI

Firefox: Tools -> page info -> security -> view cookies
Cookie policy versus same-origin policy
Cookie policy versus same-origin policy

- Consider Javascript on a page loaded from a URL U
- If a cookie is in scope for a URL U, it can be accessed by Javascript loaded on the page with URL U, unless the cookie has the httpOnly flag set.

So there isn’t exact domain match as in same-origin policy, but cookie policy instead.
Examples

**cookie 1**
- name = userid
- value = u1
- domain = login.site.com
- path = /
- non-secure

**cookie 2**
- name = userid
- value = u2
- domain = .site.com
- path = /
- non-secure

http://checkout.site.com/  
cookie: userid=u2

http://login.site.com/  
cookie: userid=u1, userid=u2

http://othersite.com/  
cookie: none

JS on each of these URLs can access the corresponding cookies even if the domains are not the same.
Indirectly bypassing same-origin policy using cookie policy

• Since the cookie policy and the same-origin policy are different, there are corner cases when one can use cookie policy to bypass same-origin policy

• Ideas how?
Victim user browser

Cookies from:
- financial.example.com
- blog.example.com

Cookies in jar with domain example.com

financial.example.com

blog.example.com

(assume attacker compromised this web server)

The browser will send the cookie for financial.example.com to blog.example.com due to domain
Victim user browser

Cookies from:

financial.example.com
(domain:financial.example.com)

blog.example.com

cookie jar

financial.example.com web server

blog.example.com web server

(assume attacker compromised this web server)

Browsers maintain a separate cookie jar per domain group, such as one jar for * example.com to avoid one domain filling up the jar and affecting another domain. Each browser decides at what granularity to hold group domains.
Victim user browser

financial.example.com web server

GET

set-cookie:

blog.example.com web server
(assume attacker compromised this web server)

Attacker sets many cookies with domain example.com which overflows the cookie jar for domain *.example.com and overwrites cookies from financial.example.com

cookie jar for *.example.com

example.com
financial.example.com
blog.example.com
example.com
Victim user browser

example.com
example.com
example.com
example.com

cookie jar for *.example.com

financial.example.com web server

blog.example.com web server

(assume attacker compromised this web server)

Attacker sets many cookies with domain example.com which overflows the cookie jar for domain *.example.com and overwrites cookies from financial.example.com
When Alice visits financial.example.com, the browser automatically attaches the attacker’s cookies due to cookie policy (the scope of the cookies is a domain suffix of financial.example.com).

Why is this a problem?
Indirectly bypassing same-origin policy using cookie policy

- Victim thus can login into attackers account at financial.example.com
- This is a problem because the victim might think its their account and might provide sensitive information
- This also bypassed same-origin policy (indirectly) because blog.example.com influenced financial.example.com
RFC6265

- For further details on cookies, checkout the standard RFC6265 “HTTP State Management Mechanism”


- Browsers are expected to implement this reference, and any differences are browser specific
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 (banks)
  – without session mgmt:
    users would have to constantly re-authenticate

• Session management:
  – Authorize user once;
  – All subsequent requests are tied to user for a period
Pre-history: HTTP auth

One username and password for a group of users

HTTP request: `GET /index.html`

HTTP response contains:

```
WWW-Authenticate: Basic realm="Password Required"
```

Browsers sends hashed password on all subsequent HTTP requests:

```
Authorization: Basic ZGFddfibzsdfgkjheczI1NXRleHQ=
```
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
Session Token Analogy

Analogy

• Show your ticket and ID
• Receive a wristband
• When you want to re-enter later, show your wristband

Actual Web

• Send your password
• Receive a session token
• When you want to make another request, send your session token
Session token

- A temporary identifier for a user, usually random or cryptographic so that an attacker cannot guess it
- If an attacker gets a session token, it could access the user’s account for the duration of that token
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 /purchase
logged-in session token

Web Site

check credentials

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

- **Browser cookie:**  
  ```
  Set-Cookie: SessionToken=fduhye63sfdb
  ```

- **Embed in all URL links:**  
  ```
  https://site.com/checkout?SessionToken=kh7y3b
  ```

- **In a hidden form field:**  
  ```
  <input type="hidden" name="sessionid" value="kh7y3b">
  ```
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
  - users might share URLs

• In a hidden form field: short sessions only

Better answer: a combination (1) and (3) above (e.g., browser cookie with CSRF protection using form secret tokens)