Cookies

CS 161 Fall 2021 - Lecture 22

Some content adapted from materials by David Wagner or Dan Boneh

Announcements

- Recording
- Homework 4 is due **Friday, October 22**, 11:59 PM PT.
- The design document draft for Project 2 is due **Friday, October 29**, 11:59 PM PT.
- We'll be holding design reviews starting next week! Signups will most likely be posted over the weekend. Start thinking about your designs early — this is a design heavy project!

Last time: SQL Injection and XSS

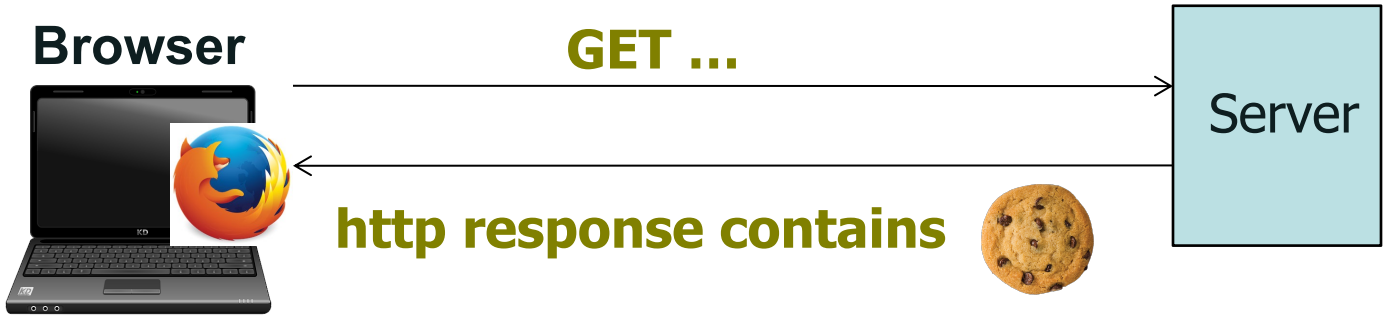
- Demo: Squigler

Cookies

- HTTP is largely stateless
- Cookies are a way to add state. This state helps link the same user's requests and helps customize websites for the user

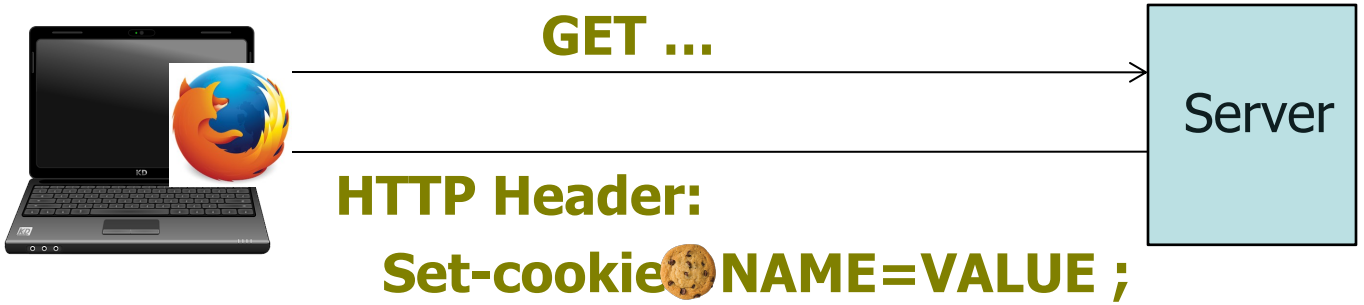
Cookies

A way of maintaining state in the browser



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

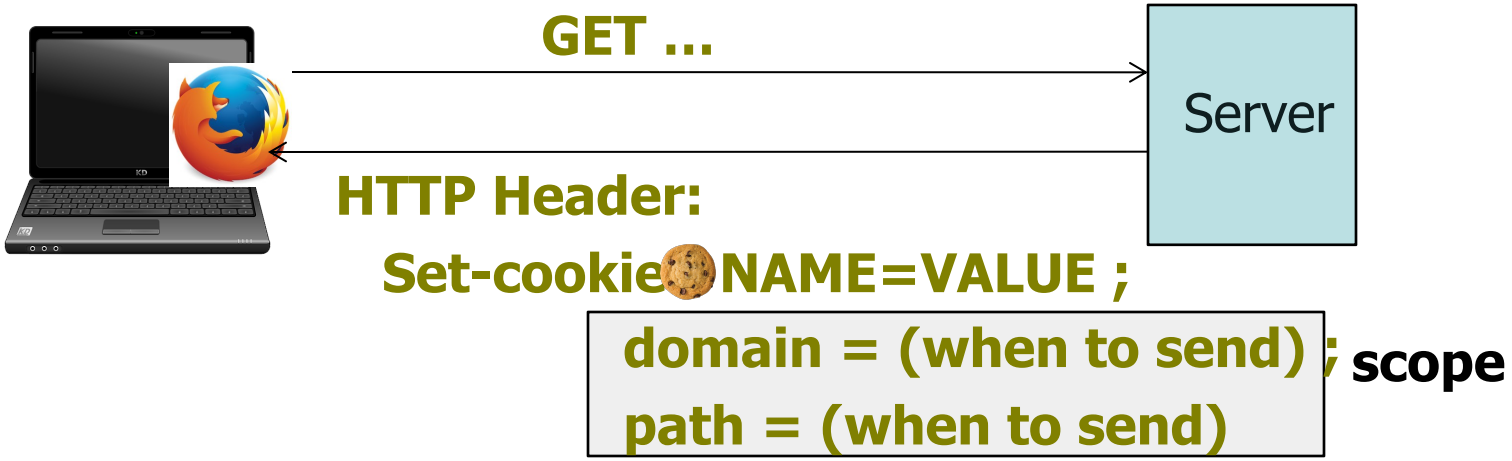
In a web console (chrome, view->developer->developer tools), type

document.cookie

to see the cookie for that site

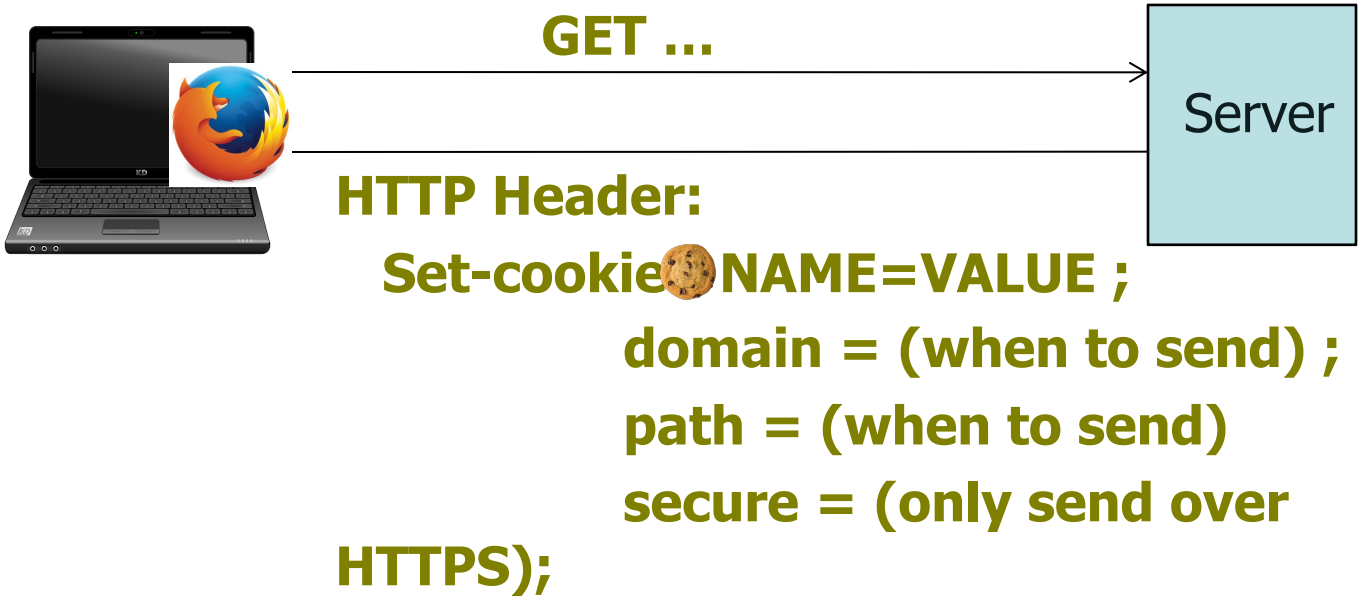
Each name=value is one cookie.
document.cookie lists all cookies in scope for document

Cookie scope



- 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

Cookie scope



- **Secure: sent over https only**
  - **https provides secure communication using TLS (encryption and authentication)**

Cookie scope

GET …

Server

HTTP Header:

Set-cookie NAME=VALUE ;

domain = (when to send) ; scope

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"

[top-level domains, e.g. '.com']

allowed domains	disallowed domains
login.site.com	user.site.com
.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:

domain	Whether it will be set
(value omitted)	*foo.example.com* (exact)
bar.foo.example.com	
foo.example.com	
baz.example.com	
example.com	yes
ample.com	
.com	

When browser sends cookie

Goal: server only sees cookies in its scope

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”]

When browser sends cookie

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:

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

Credits: The Tangled Web: A Guide to Securing Modern Web Applications, by Michał Zalewski

Examples

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

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

Examples

cookie 1
name = userid
value = u1
domain = login.site.com
path = /
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=u2**

https://login.site.com/ **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)

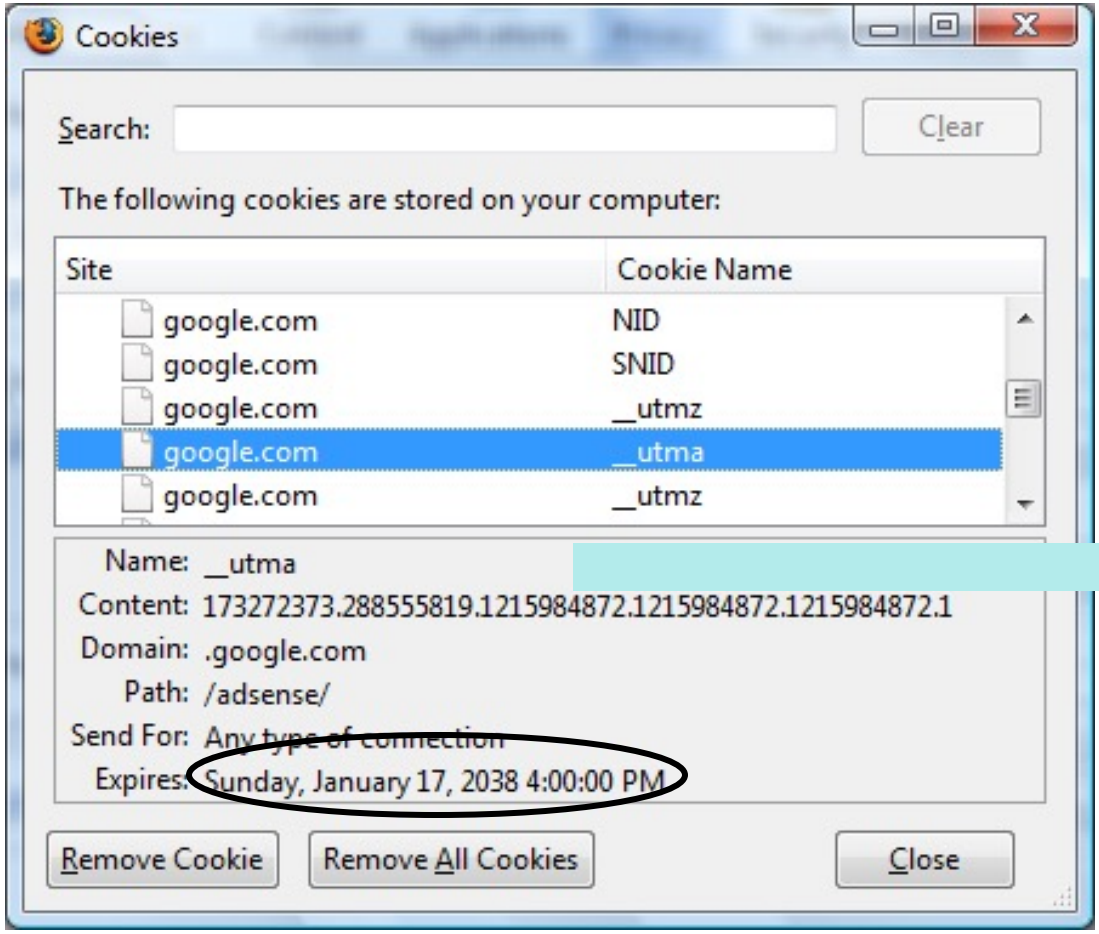
- Deleting a cookie:

  document.cookie = “name=; expires= Thu, 01-Jan-00”

document.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 the cookie policy is invoked 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

RFC6265

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

https://tools.ietf.org/html/rfc6265

- Browsers are expected to implement this reference, and any differences are browser specific