1

Cookies

East Tennessee State University
Department of Computing
CSCI 1720 – Intermediate Web

2

First

Understanding Stateful vs. Stateless applications

3

'State-ful' vs. 'State-less'

The state of an application (or anything else, really) is its condition or quality of being at a given moment in time--its state of being

Whether something is stateful or stateless depends on how long the state of interaction with it is being recorded and how that information needs to be stored

4

'State-ful' vs. 'State-less'

Stateless

A stateless process or application can be understood in isolation

There is no stored knowledge of or reference to past transactions

Each transaction is made as if from scratch for the first time

Stateless applications provide one service or function and use content delivery network (CDN), web, or print servers to process these short-term requests

5

'State-ful' vs. 'State-less'

Stateless

An example of a stateless transaction would be doing a search online to answer a question you've thought of

You type your question into a search engine and hit enter

If your transaction is interrupted or closed accidentally, you just start a new one

Think of stateless transactions as a vending machine: a single request and a response

6

'State-ful' vs. 'State-less'

Stateful

Stateful applications and processes are those that can be returned to again and again, like online banking or email

They're performed with the context of previous transactions and the current transaction may be affected by what happened during previous transactions

For these reasons, stateful apps use the same servers each time they process a request from a user

7

'State-ful' vs. 'State-less'

Stateful

If a stateful transaction is interrupted, the context and history have been stored so you can more or less pick up where you left off

Stateful apps track things like window location, setting preferences, and recent activity

You can think of stateful transactions as an ongoing periodic conversation with the same person

8

'State-ful' vs. 'State-less'

Stateful

The majority of applications we use day to day are stateful, but as technology advances, microservices and containers make it easier to build and deploy applications in the cloud

9

Now…Cookies

What are they and why should we use them?

10

Cookies – A Brief History

HTTP, or the Hypertext Transfer Protocol, is a stateless protocol

According to Wikipedia, it's a stateless protocol because it "does not require the HTTP server to retain information or status about each user for the duration of multiple requests"

You can still see this today with simple websites – you type in the URL to the browser, the browser makes a request to a server somewhere, and the server returns the files to render the page and the connection is closed

11

Cookies – A Brief History

Now imagine that you need to sign in to a website to see certain content, like with LinkedIn

The process is largely the same as the one above, but you're presented with a form to enter in your email address and password

You enter that information in and your browser sends it to the server

The server checks your login information, and if everything looks good, it sends the data needed to render the page back to your browser

12

Cookies – A Brief History

But if LinkedIn was truly stateless, once you navigate to a different page, the server would not remember that you just signed in

It would ask you to enter in your email address and password again, check them, then send over the data to render the new page

That would be super frustrating, wouldn't it? A lot of developers thought so, too, and found different ways to create stateful sessions on the web

13

The Invention of the HTTP Cookie

Lou Montoulli, a developer at Netscape in the early 90s, had a problem – he was developing an online store for another company, MCI, which would store the items in each customer's cart on its servers

This meant that people had to create an account first, it was slow, and it took up a lot of storage

MCI requested for all of this data to be stored on each customer's own computer instead

Also, they wanted everything to work without customers having to sign in first

14

The Invention of the HTTP Cookie

To solve this, Lou turned to an idea that was already pretty well known among programmers: the magic cookie

A magic cookie, or just cookie, is a bit of data that's passed between two computer programs

They're "magic" because the data in the cookie is often a random key or token, and is really just meant for the software using it

Lou took the magic cookie concept and applied it to the online store, and later to browsers as a whole

15

The Invention of the HTTP Cookie

Now that you know about their history, let's take a quick look at how cookies are used to create stateful sessions on the web

16

How Cookies Work

One way to think of cookies is that they're a bit like the wristbands you get when you visit an amusement park

For example, when you sign in to a website, it's like the process of entering an amusement park

First you pay for a ticket, then when you enter the park, the staff checks your ticket and gives you a wristband

This is like how you sign in – the server checks your username and password, creates and stores a session, generates a unique session id, and sends back a cookie with the session id

17

How Cookies Work

(Note that the session id is not your password, but is something completely separate and generated on the fly)

While you're in the amusement park, you can go on any ride by showing your wristband

Similarly, when you make requests to the website you're signed in to, your browser sends your cookie with your session id back to the server. The server checks for your session using your session id, then returns data for your request

Finally, once you leave the amusement park, your wristband no longer works – you can't use it to get back into the park or go on more rides

18

How Cookies Work

This is like signing out of a website

Your browser sends your sign out request to the server with your cookie, the server removes your session, and lets your browser know to remove your session id cookie

If you want to get back into the amusement park, you'd have to buy another ticket and get another wristband

In other words, if you want to continue using the website, you'd have to sign back in

How to use Cookies - Limitations

Cookies are quite limited compared to some modern alternatives to storing data in the browser like `localStorage` or `sessionStorage`

Here's a rundown of cookies compared to those other technologies

How to use Cookies - Limitations

	COOKIES	LOCAL STORAGE	SESSION STORAGE
Capacity	4KB	10MB	5MB
Accessible from	Any window	Any window	Same tab
Expires	Manually set	Never	On tab close
Storage location	Browser and server	Browser only	Browser only
Sent with requests	Yes	No	No

22

How to use Cookies - Limitations

Cookies are an older technology, and have a very limited capacity

Still, there's quite a bit you can do with them

And their small size makes it easy for the browser to send cookies with each request to the server

It's also worth mentioning that browsers only allow cookies to work from one domain for security reasons

23

How to use Cookies - Limitations

So if you sign in to your bank at, say, ally.com, then cookies will only work within that domain and its subdomains

For example, your ally.com cookie will work on ally.com, ally.com/about, and the subdomain www.ally.com, but not axos.com.

This means that, even if you have accounts and are signed in at both ally.com and axos.com, those sites won't be able to read each other's cookies

24

How to use Cookies - Limitations

It's important to remember that your cookies are sent with every request you make in the browser

This is very convenient, but has some serious security implications

Just remember that cookies are meant to be openly read and sent, so you should never store sensitive information like passwords in them

25

How to Set a Cookie in JavaScript

Cookies are really just strings with key / value pairs

Though you'll probably work with cookies more on the backend, there may be times you'll want to set a cookie on the client side

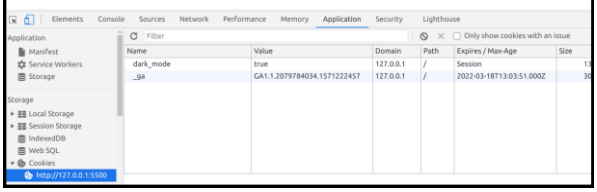
Here's how to set a cookie in vanilla JavaScript:

```
document.cookie = 'dark_mode=true'
```


26

How to Set a Cookie in JavaScript

Then when you open the developer console, click "Application" and then on the site under "Cookies", you'll see the cookie you just added



How to Set a Cookie in JavaScript

If you take a closer look at your cookie, you'll see that its expiration date is set to Session

Expires / Max-Age
Session

That means the cookie will be destroyed when you close your tab / browser

That might be the behavior you want, like for an online store with payment information

But if you want your cookie to last longer, you'll need to set an expiration date

28

How to set an Expiration Date

```
// expires 1 week from now
document.cookie = 'dark_mode=true; expires=Fri,
26 Feb 2021 00:00:00 GMT'
```

Expires / Max-Age
2021-02-26T00:00:00.000Z

29

How to set an Expiration Date

Or you could use the max-age attribute with the number of seconds you'd like your cookie to last:

```
// expires 1 week from now
document.cookie = 'dark_mode=true; max-
age=604800';
```


30

How to Update a Cookie in JavaScript

Whether or not your cookie has an expiration date, updating it is easy

Just change the value for your cookie, and the browser will automatically pick it up:

```
// expires 1 week from now
document.cookie = "dark_mode=false; max-age=604800";
```


31

How to set the path for a cookie in JavaScript

Sometimes you'll only want your cookie to work with certain parts of your website

Depending on how your website is set up, one way to do this is with the path attribute

Here's how to make it so a cookie only works on the about page at /about:

```
document.cookie = 'dark_mode=true; path=/about';
```


32

How to Set the Path for a Cookie in JavaScript

```
document.cookie = 'dark_mode=true; path=/about';
```

Now your cookie will only work on `/about` and other nested subdirectories like `/about/team`, but not on `/blog`

33

How to delete a cookie in JavaScript

To delete a cookie in JavaScript, just set the expires attribute to a date that's already passed:

```
// 1 week earlier
document.cookie = 'dark_mode=true; expires=Sun,
14 Feb 2021 00:00:00 GMT';
```

You could also use max-age and pass it a negative value:

```
// 1 minute earlier
document.cookie = 'dark_mode=true; max-age=-60';
```


34

Security Concerns with Cookies

In general, cookies are very secure when implemented correctly

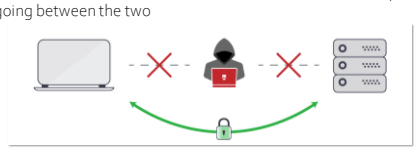
Browsers have a lot of built-in limitations that we covered earlier, partly due to the age of the technology, but also to improve security

Still, there are a few ways that a bad actor can steal your cookie and use it to wreak havoc

35

Man-in-the-middle Attacks

A man-in-the-middle (MitM) attack describes a broad category of attacks where an attacker sits between a client and a server and intercepts the data going between the two



36

Man-in-the-middle Attacks

This can be done in a lot of ways:

- Gaining access to or listening in on an insecure website
- Mimicking a public WiFi router
- DNS spoofing
- Malware / adware

37

Man-in-the-middle Attacks

38

Man-in-the-middle Attacks

As a developer, you can greatly reduce the chance of a MitM attack by ensuring that you enable HTTPS on your server, use an SSL certificate from a trusted certificate authority, and ensure your code uses HTTPS instead of the insecure HTTP

In terms of cookies, you should add the Secure attribute to your cookies so they can only be sent over a secure HTTPS connection:

```
document.cookie = 'dark_mode=false; Secure';
```


39

Man-in-the-middle Attacks

The Secure attribute doesn't actually encrypt any data in your cookie – it just ensures that the cookie can't be sent over an HTTP connection

However, a bad actor could still possibly intercept and manipulate the cookie

To prevent this from happening, you can also use the HttpOnly parameter:

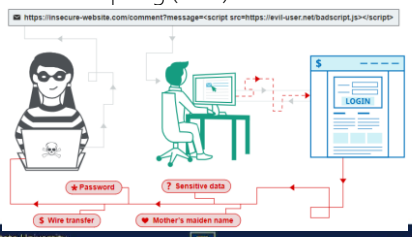
```
document.cookie = 'dark_mode=false; Secure; HttpOnly';
```


Cross-Site Scripting (XSS) Attacks

An XSS (cross-site scripting) attack describes a category of attacks when a bad actor injects unintended, potentially dangerous code into a website

These attacks are very problematic because they could affect every person that visits the site

Cross-Site Scripting (XSS) Attacks



Cross-Site Scripting (XSS) Attacks

If a site has a comments section and someone is able to include malicious code as a comment, it's possible that every person who visits the site and reads that comment will be affected

In terms of cookies, if a bad actor pulls off a successful XSS attack on a site, they could gain access to session cookies and access the site as another signed in user

From there, they may be able to access the other user's settings, buy things as that user and have it shipped to another address, and so on

43

Cross-Site Scripting (XSS) Attacks

44

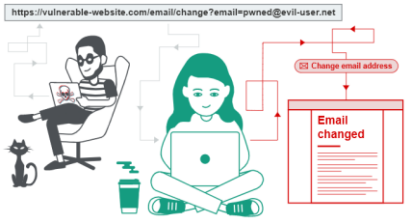
Cross-Site Request Forgery (CSRF) Attacks

A CSRF (cross-site request forgery) attack is when a bad actor tricks a person into carrying out an unintended, potentially malicious action

For example, if you're signed into a site and click on a link in a comment, if that link is part of a CSRF attack, it may lead to you unintentionally changing your sign in details, or even deleting your account

45

Cross-Site Request Forgery (CSRF) Attacks



Cross-Site Request Forgery (CSRF) Attacks

While CSRF attacks are somewhat related to XSS attacks, specifically reflected XSS attacks where someone inserts malicious code into a site, each preys on a different type of trust

According to Wikipedia, while XSS "exploits the trust a user has for a particular site, CSRF exploits the trust that a site has in a user's browser."

Cross-Site Request Forgery (CSRF) Attacks

Cross-Site Request Forgery (CSRF) Attacks

As for cookies, one way to prevent possible CSRF attacks is with the SameSite flag:

```
document.cookie = 'dark_mode=false; Secure;
HttpOnly; SameSite=Strict';
```


Cross-Site Request Forgery (CSRF) Attacks

There are a few values you can set for SameSite:

Lax: Cookies are not sent for embedded content (images, iframes, etc.) but are sent when you click on a link or send a request to the origin the cookie is set for. For example, if you're on testing.com and you click on a link to go to test.com/about, your browser will send your cookie for test.com with that request

Strict: Cookies are only sent when you click on a link or send a request from the origin the cookie is set for. For example, your test.com cookie will only be sent while you're in and around test.com, and not coming from other sites like testing.com

Cross-Site Request Forgery (CSRF) Attacks

There are a few values you can set for SameSite:

None: Cookies will be sent with every request, regardless of context. If you set SameSite to None, you must also add the Secure attribute. It's better to avoid this value if possible

Major browsers handle SameSite a bit differently. For example, if SameSite isn't set on a cookie, Google Chrome sets it to Lax by default

Alternatives to Cookies

You might be wondering, if there are so many potential security flaws with cookies, why are we still using them? Surely there must be a better alternative

These days, you can use either `sessionStorage` or `localStorage` to store information that originally used cookies

For stateful sessions, there's token-based authentication with things like JWT (JSON Web Tokens)

Alternatives to Cookies

While it may seem like you have to choose between cookie-based or token-based authentication, it's possible to use both

For example, you might want use cookie-based authentication when someone signs in through the browser, and token-based authentication when someone signs in through a phone app

To muddy the waters a bit more, authentication-as-a-service providers like Autho allow you to do both types of authentication

Cookies

Despite all of the issues surrounding them, cookies are such an important part of web development that, should they disappear without a replacement, many of our favorite web applications would be rendered useless

Cookie Questions?

55

55

Sources:

- https://www.redhat.com/en/topics/cloud-native-apps/stateful-vs-stateless
- https://www.freecodecamp.org/news/everything-you-need-to-know-about-cookies-for-web-development/
- https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
- https://www.youtube.com/watch?v=AwicscsvGLg
- https://www.netsparker.com/blog/web-security/man-in-the-middle-attack-how-avoid/
- https://portswigger.net/web-security/cross-site-scripting
- https://portswigger.net/web-security/csrf

56

Copyrights

Presentation prepared by and copyright of John Ramsey, East Tennessee State University, Department of Computing . (ramseyjw@etsu.edu)

57