Web Headers and Cookies

Alex Curtiss | CSCI 3403

Patch Notes

Last week’s weekly exercise was due yesterday, this week’s is available on Canvas

Python

Python data structures:

```python
# List (array)
users = ["alice", "bob", "carol"]
print(users[1]) # Prints "bob"

# Dictionary (hashtable)
user_passwords = {
    "alice": "swordfish",
    "bob": "hunter2",
    "carol": "12345"
}
print(user_passwords["carol"]) # Prints "12345"
```

Web Fundamentals

Recap:

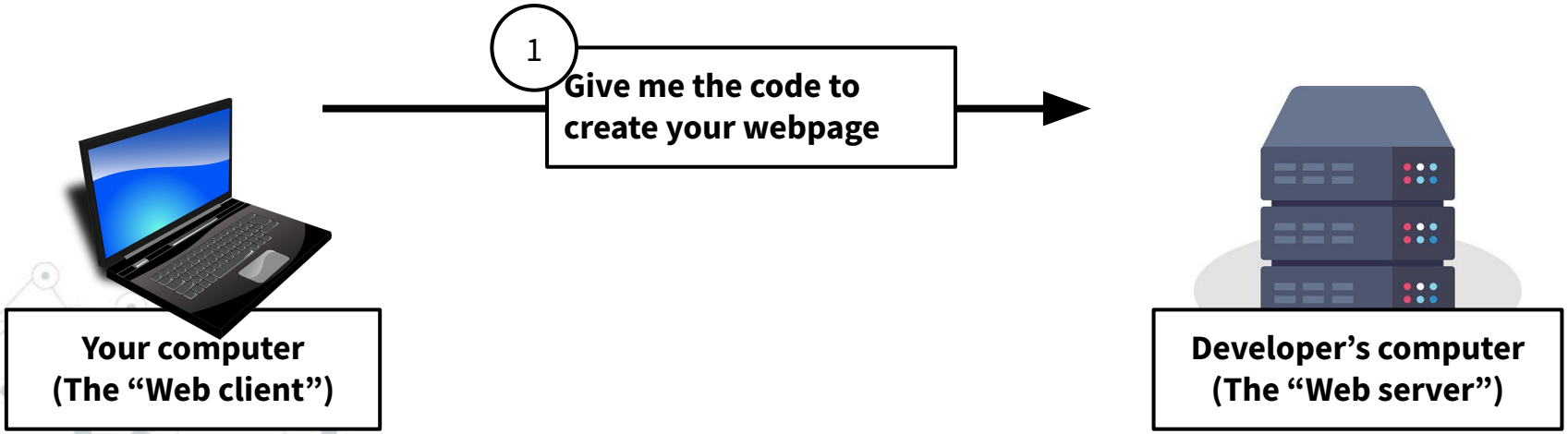
- Client code: Displays the webpage. Written in HTML.
- Server code: Responds to messages from the client.

Your computer
(The “Web client”)

Developer’s computer
(The “Web server”)

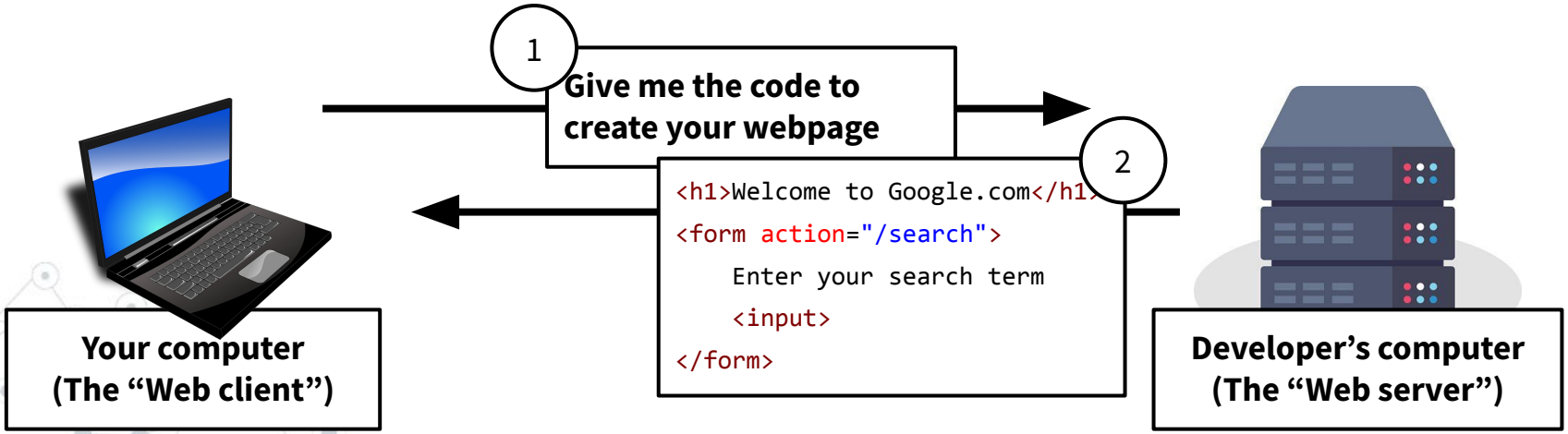
Web Fundamentals

1. Your computer (the “client”) asks for the website from the web developer’s computer (the “server”)



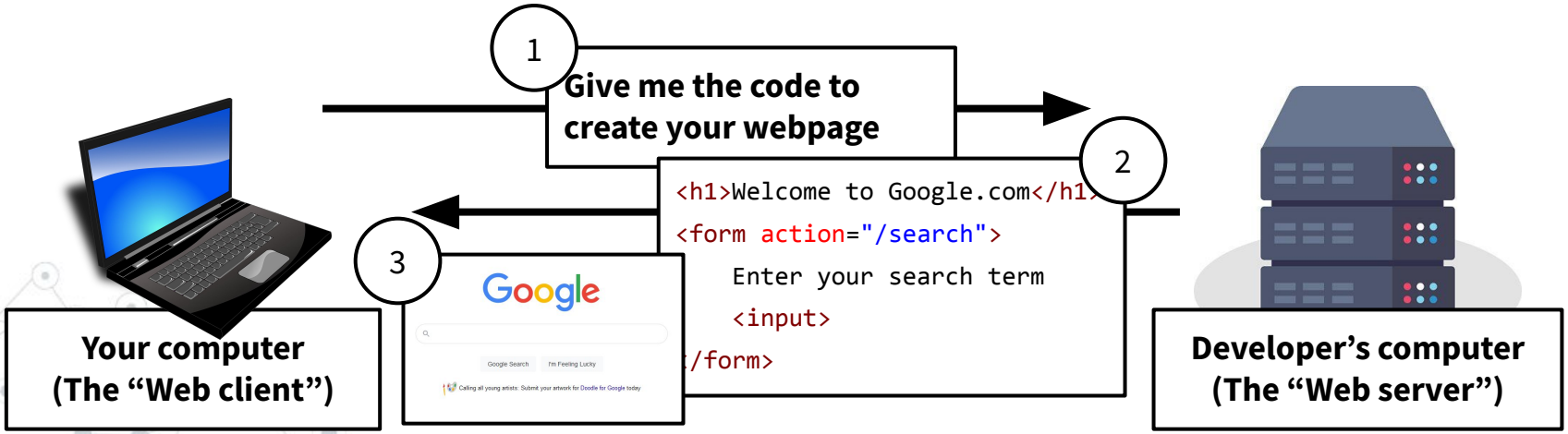
Web Fundamentals

1. Your computer (the “client”) asks for the website from the web developer’s computer (the “server”)
2. The server sends the code needed to create the website



Web Fundamentals

1. Your computer (the “client”) asks for the website from the web developer’s computer (the “server”)
2. The server sends the code needed to create the website
3. The client runs the code to display the website



Web Fundamentals

Web request: A message sent from the client to the server, such as asking for the web page.

HTTP

Web clients and server use a specific format, called **HyperText Transfer Protocol (HTTP)**.

HTTP

Path

```
GET /home/index.html HTTP/1.1
Host: www.google.com
User-Agent: Firefox 96.0.3
```

Your computer
(The “Web client”)

Developer’s computer
(The “Web server”)

HTTP

Method

- **GET: Get data**
- **POST: Send data**

Path

```
GET /home/index.html HTTP/1.1
Host: www.google.com
User-Agent: Firefox 96.0.3
```

Your computer
(The “Web client”)

Developer’s computer
(The “Web server”)

HTTP

Method

- **GET: Get data**
- **POST: Send data**

Path

```
POST /login HTTP/1.1
Host: www.google.com
User-Agent: Firefox 96.0.3

username:alex
```

Data (POST only)

Your computer
(The “Web client”)

Developer’s computer
(The “Web server”)

HTTP

```
<form action="/login" method="post">
    <input type="text" name="username" value="alex">
</form>
```

```
POST /login HTTP/1.1
Host: www.google.com
User-Agent: Firefox 96.0.3

username:alex
```

HTTP

Headers: Optional metadata about the request, e.g.

- **User-Agent:** Which browser and OS you are running
- **Referer:** The previous website which linked to this one

```
POST /login HTTP/1.1
Host: www.google.com
User-Agent: Firefox 96.0.3

username:alex
```

Headers

HTTP

The response from the server contains a “status”, data (often HTML), and different headers than the request, e.g.

- **Server:** The server language or framework

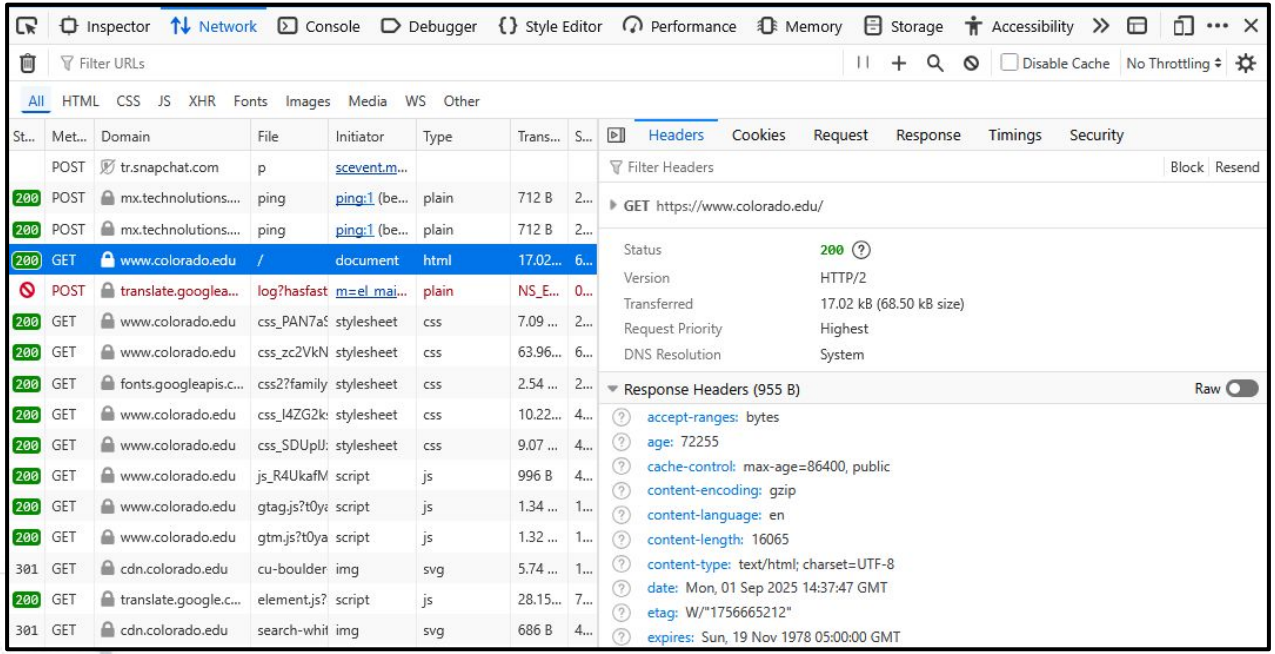
```
HTTP/1.1 200 OK
Server: apache/2.1

<h1>My website</h1>
<p>...</p>
```

Response status

Data (typically HTML)

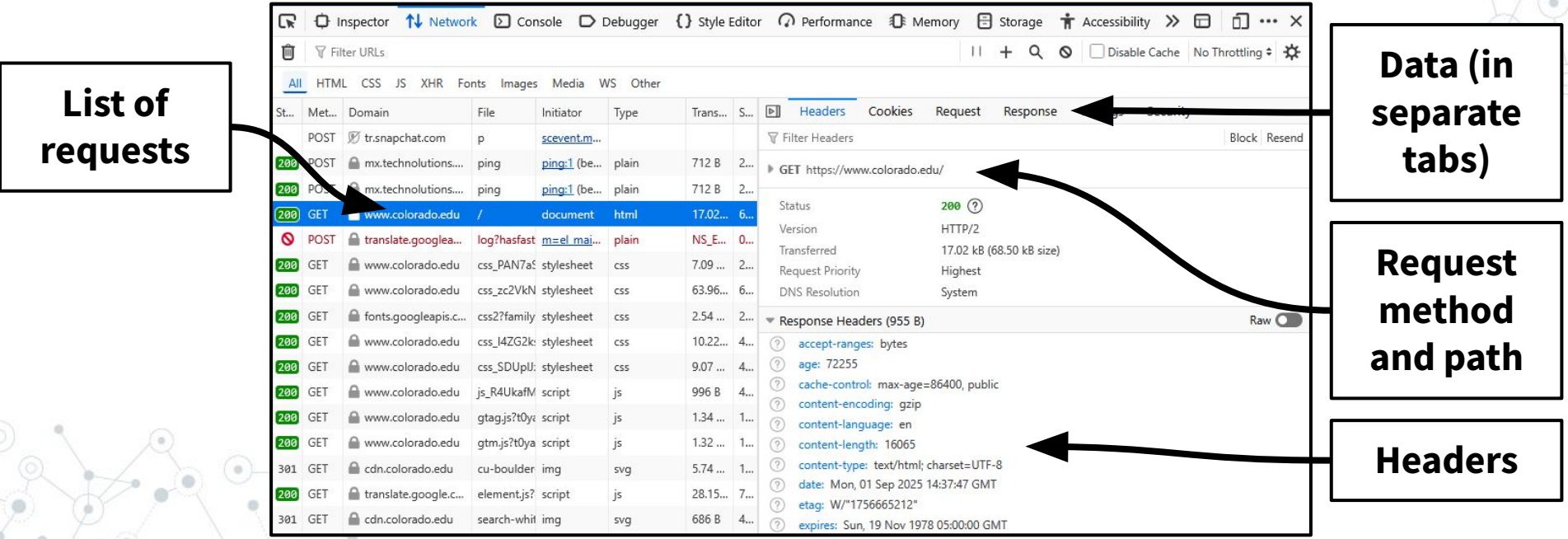
HTTP

The developer tools “network” tab shows HTTP requests:



HTTP

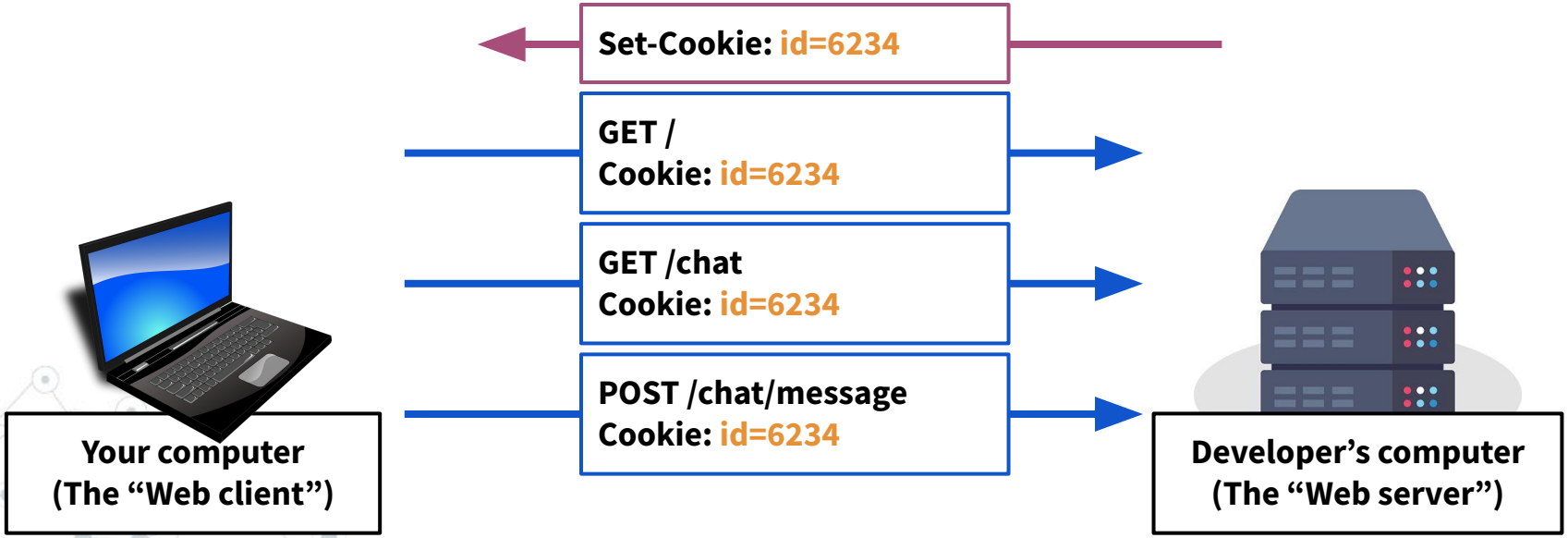
The developer tools “network” tab shows HTTP requests:



Cookies

Cookies

Cookies: Header values which are saved by the browser and sent back to the server with each request



[Demo: Login with cookies]

Server-side code

Login with cookies demo code:

```python
from flask import Flask, redirect, render_template, request

app = Flask(__name__)

@app.route("/")
def index():
    current_user = request.cookies.get("logged_in_as")
    return render_template("index.html", current_user=current_user)

@app.route("/login", methods=["GET"])
def login_get():
    return render_template("login.html")

@app.route("/login", methods=["POST"])
def login_post():
    username = request.form["username"]
    password = request.form["password"]

    if username == "alex" and password == "swordfish":
        response = redirect("/")
        response.set_cookie("logged_in_as", "alex")
        return response

app.run(debug=True)
```

```html
<!-- index.html -->
<h1>Example Site</h1>

<p>Logged in as: {{ user }}</p>

<a href="/login">Login</a>

<!-- login.html -->
<h1>Example Site</h1>

<form action="/login" method="post">
    Username:
    <input type="text" name="username">

    Password:
    <input type="password" name="password">

    <input type="submit">
</form>

<form action="/login" method="post">
    <input type="text" name="username">
</form>
```

What is wrong with that approach?

Cookies

Cookies are often used for authentication. This makes them just as valuable as passwords!

Cookies

Cookies are often used for authentication. This makes them just as valuable as passwords!

- **Terribly insecure:** Cookie is just the username
- **Better:** Cookie is a long, random string and the server keeps a mapping of cookie/username values

```
{
    "7ce5a141-6431-4d83-8913-25447d35e7a0": "alex",
    "d4622f93-59b2-4a49-a104-39af77a01cc0": "admin",
}
```

[Demo: Secure cookies]

Cookies

Secure cookies demo code:

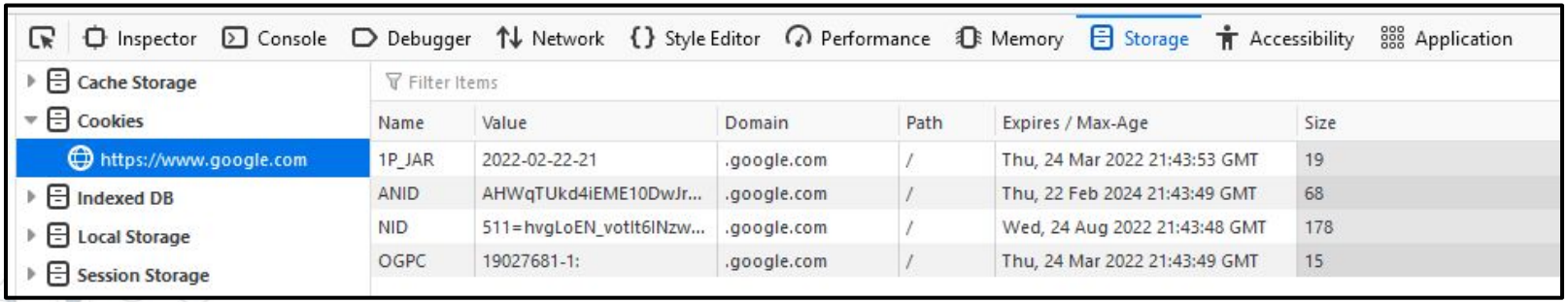
```python
import random
from flask import Flask, redirect, render_template, request

app = Flask(__name__)

cookie_map = {}

@app.route("/")
def index():
    cookie = request.cookies.get("logged_in_as")
    current_user = cookie_map.get(cookie)

    return render_template("index.html", current_user=current_user)

@app.route("/login", methods=["GET"])
def login_get():
    return render_template("login.html")

@app.route("/login", methods=["POST"])
def login_post():
    username = request.form["username"]
    password = request.form["password"]

    if username == "alex" and password == "swordfish":
        cookie = random.randbytes(20).hex()
        cookie_map[cookie] = "alex"

        response = redirect("/")
        response.set_cookie("logged_in_as", cookie)
        return response

app.run(debug=True)
```

```html
<!-- index.html -->
<h1>Example Site</h1>

<p>Logged in as: {{ user }}</p>

<a href="/login">Login</a>

<!-- login.html -->
<h1>Example Site</h1>

<form action="/login" method="post">
    Username:
    <input type="text" name="username">

    Password:
    <input type="password" name="password">

    <input type="submit">
</form>

<form action="/login" method="post">
    <input type="text" name="username">
</form>
```

Cookies

Even if cookies are long and random, they can still be stolen off either the client or the server.

- **Example:** Somebody with physical access to your computer can open the cookie menu and read them



[Demo: Stealing a cookie]

The lecture is over go home