TK2100: Informasjonssikkerhet Lesson 09: HTTPS and Sessions

Dr. Andrea Arcuri
Westerdals Oslo ACT
University of Luxembourg

Goals

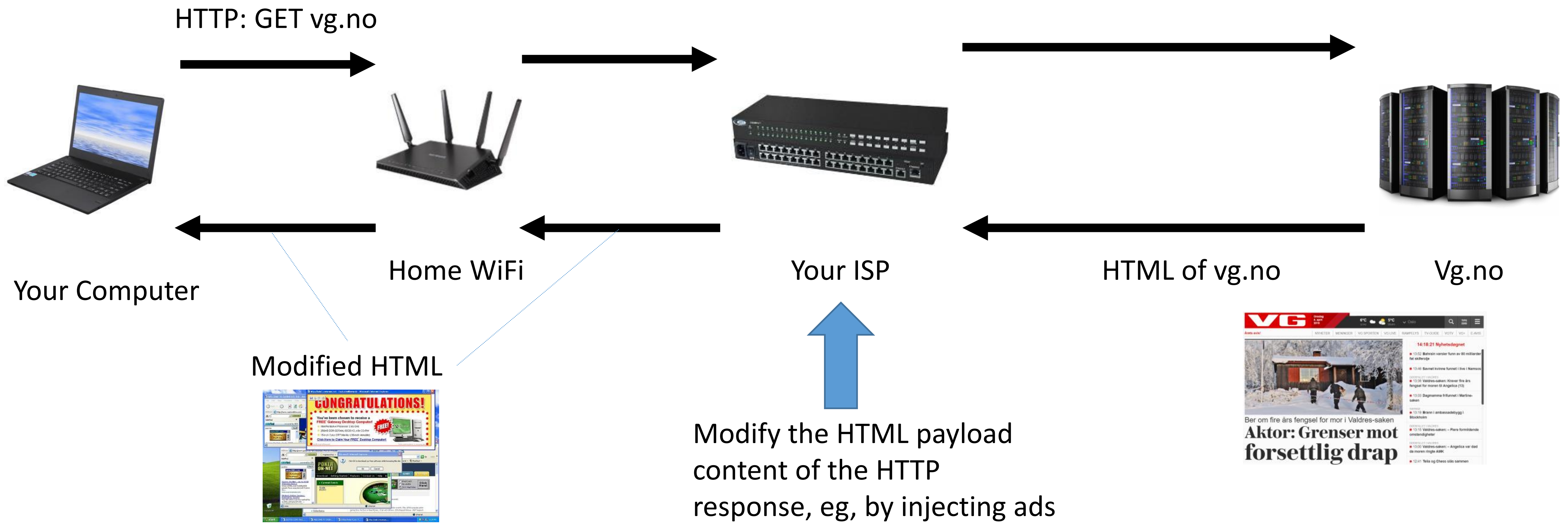
- Understand the bases of HTTPS, and why it should always be used instead of HTTP
- Understand the bases of HTTP Cookies and handling of sessions in HTTP/S
- Learn how to carry out cross-site request attacks

HTTPS

HTTP is Not Secure

- Messages in HTTP are not secure, because not encrypted
- HTTPS extends HTTP by using encryption on all messages
- Important to do for *ALL* kinds of communications, even if not critical
  - Not just for authentication (login) in banks or other internet services
- Example, ISPs (eg Comcast) can inject ads in web pages
  - ISP -> Internet Service Provider

- What would be the point of encrypting pages of a newspaper?
- If not encrypted (ie HTTP instead of HTTPS) anyone between you and the target server can alter the payload of the messages…



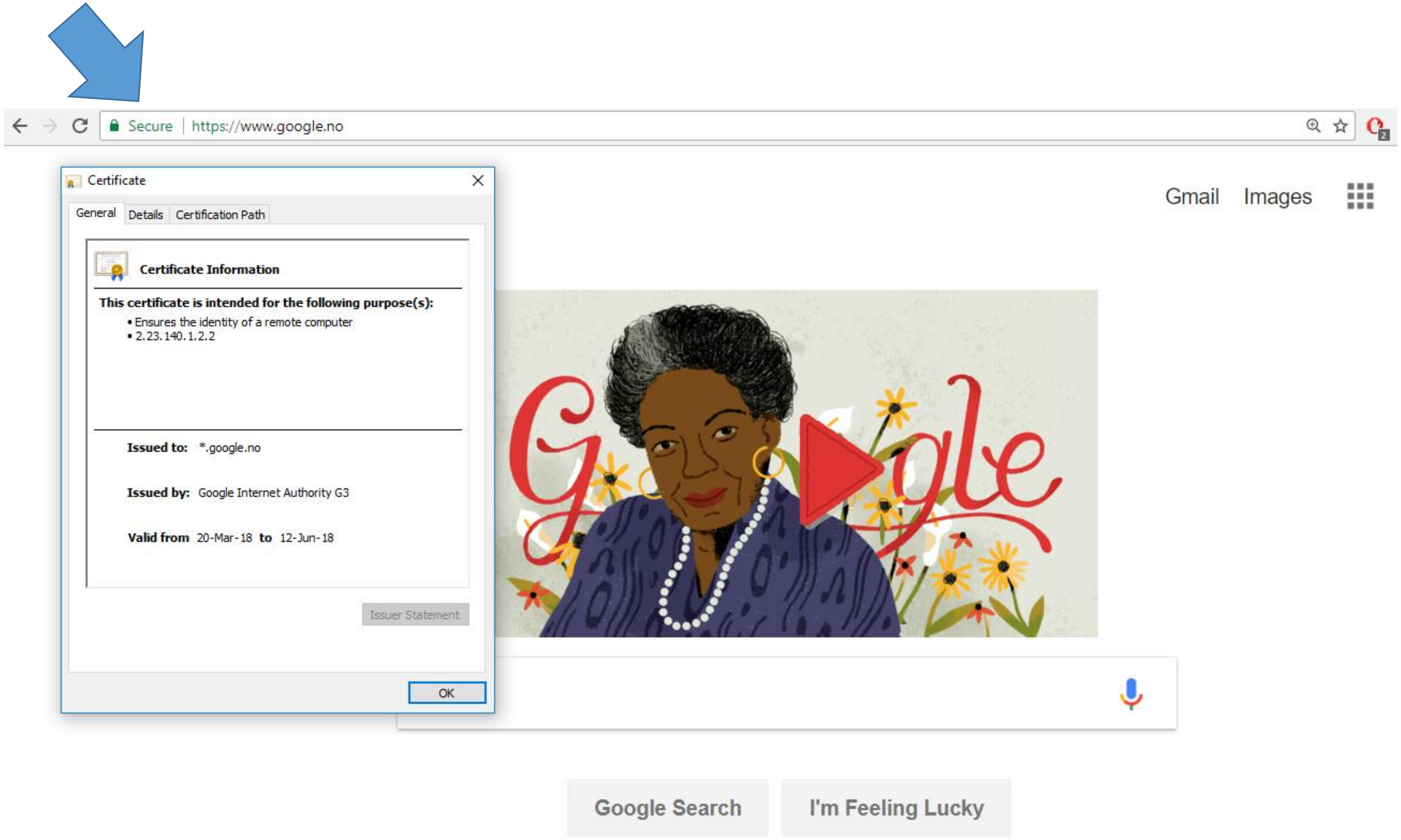
Modify the HTML payload content of the HTTP response, eg, by injecting ads

TLS/SSL Digital Certificates

- When using HTTPS to a server, first download a digital certificate
- Digital Certificate (DC) contains:
  1. domain name (eg. www.facebook.com)
  2. trusted certificate authority (CA)
  3. server's *public encryption key*, ie RSA algorithm
- Certificates have expiration time
- The certificates themselves are signed, with the OS having public keys to check their integrity , ie they are signed with the private keys of the CA

Cont.

- Eve would not be able to forge a DC for *www.facebook.com* (eg needed for her phishing attacks), because she would need to sign it with the private key of a CA
- Note: this is secure only as long as one can *trust* the CAs, but some of those have been compromised in the past…



Certificate

General | Details | Certification Path

Certificate Information

This certificate is intended for the following purpose(s):

- Ensures the identity of a remote computer
- 2.23.140.1.2.2

Issued to: *.google.no

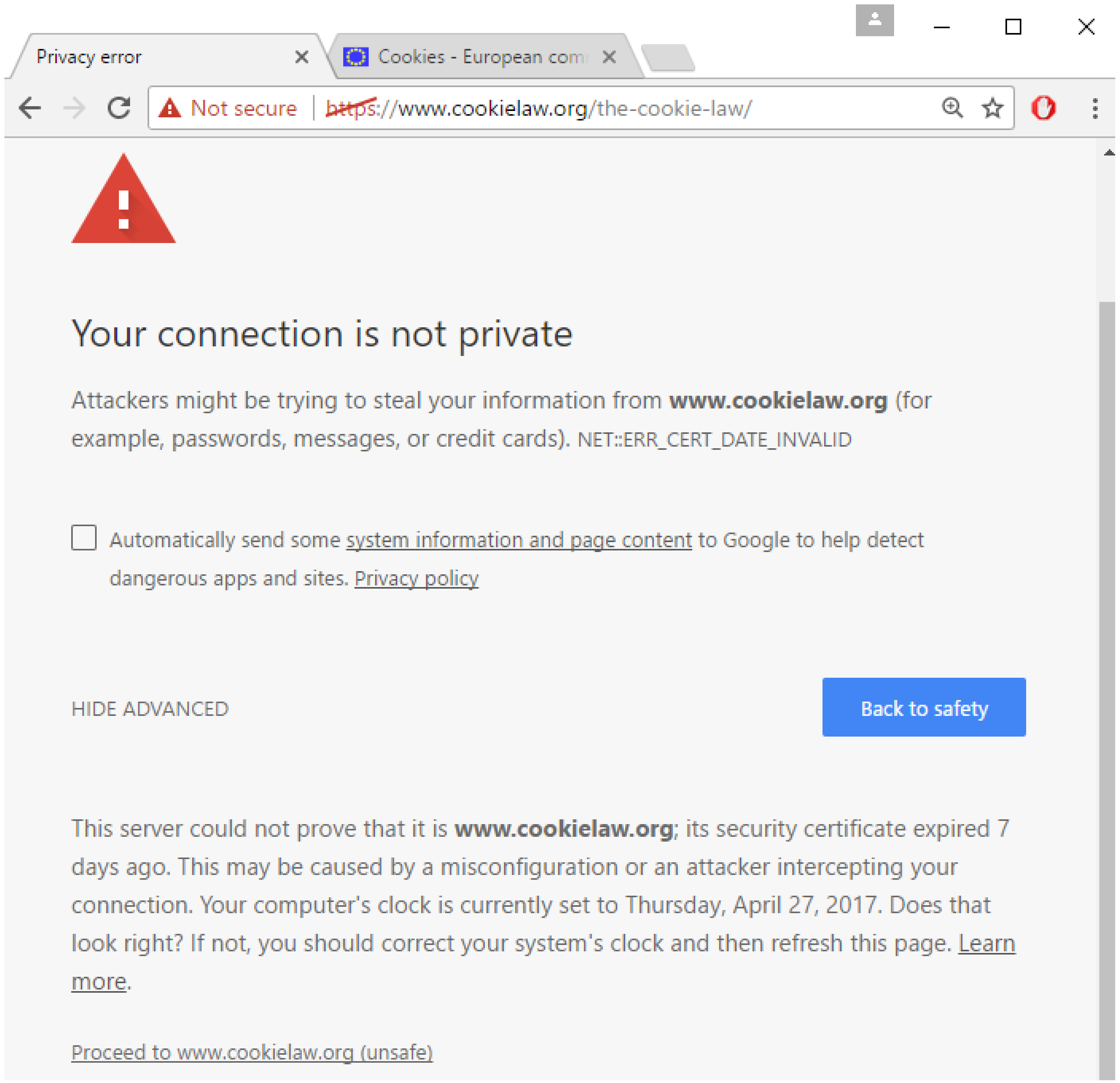
Issued by: Google Internet Authority G3

Valid from 20-Mar-18 **to** 12-Jun-18

Issuer Statement

OK

What if browser detects that the certificate is invalid?



Your connection is not private

Attackers might be trying to steal your information from **www.cookielaw.org** (for example, passwords, messages, or credit cards). NET::ERR_CERT_DATE_INVALID

Automatically send some system information and page content to Google to help detect dangerous apps and sites. Privacy policy

This server could not prove that it is **www.cookielaw.org**; its security certificate expired 7 days ago. This may be caused by a misconfiguration or an attacker intercepting your connection. Your computer's clock is currently set to Thursday, April 27, 2017. Does that look right? If not, you should correct your system's clock and then refresh this page. Learn more.

Proceed to www.cookielaw.org (unsafe)

If Certification Fails

- 2 main possibilities
- 1) Expired certificate: developers have not renewed their certificates with the CA
- 2) Man-in-the-middle attack
  - Trying to pretend to be the web app you want to connect to, so can steal your login/password, eg by serving a fake login page that is equal to the original one
  - Easy way to do such attack? Use a router, have an open WiFi called “Free WiFi”, go close to a bar/restaurant, and wait for people to connect to it, give them fake pages with fake certificates, and wait for those people to click “Proceed” when browser complains about certificate is invalid
  - **NEVER PRESS “PROCEED” WITH INVALID CERTIFICATE ON UNTRUSTED NETWORK!!!** And if you really have to, do not provide any sensitive information, eg passwords, although it would be still a problem with cookies…

Login

Authentication/Authorization

- **Authentication**:
  - do I know who a user X is?
  - how to distinguish X from a different user Y?
- **Authorization**:
  - once I know that the current user is X, what is X allowed to do?
  - can s/he delete data?
  - can s/he see data of other users?
  - etc.
- Of course, they only make sense with encryption (eg HTTPS), so no one can decode and tamper with the messages…

Authentication/Authorization failures

- If not authenticated, server can:
  - redirect to login page, HTTP status code 3xx
  - error page, HTTP status 401 *Unauthorized*
- If authenticated but not authorized
  - eg user X tries to access data of Y
  - 3xx redirection
  - HTTP status 403 *Forbidden*

Blacklisting vs Whitelisting

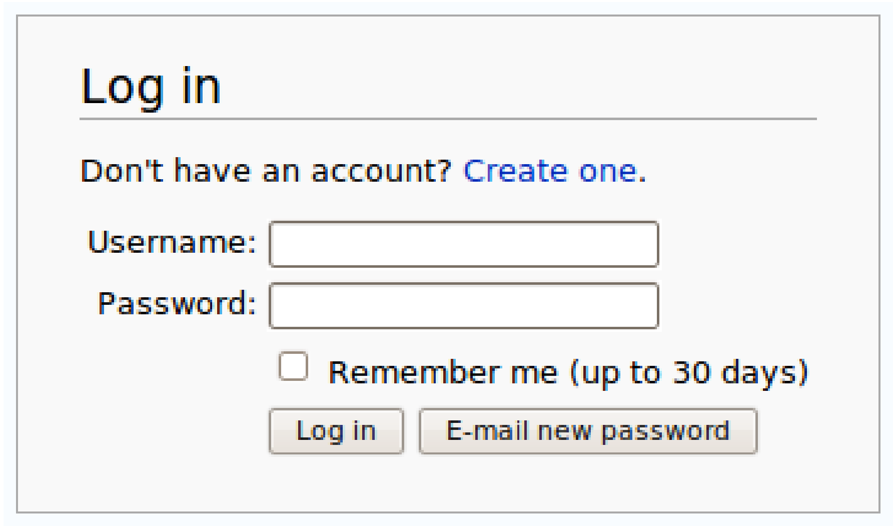
- Authorization is done on the server, and will depend on the language/framework
  - JEE, Spring, PHP, .Net, NodeJS, etc.
  - user will just get either a 3xx or 403 response
- *Blacklisting*: everything is allowed by default. What is not allowed for a given user/group has to be explicitly stated
  - Usually not a good idea, as easy to forget to blacklist some critical operation
- *Whitelisting*: nothing is allowed by default. What is allowed has to be explicitly stated
  - “forgetting to allow something” (reduced functionality) is much, much better than “forgetting to forbid something” (security problem)

Authentication: first steps

- Server does not know who the user is
- Server only sees incoming HTTP/S messages
  - not necessarily from a browser… user can do direct TCP connections from scripts
- HTTP/S is stateless
- Need a way to tell that sequence of HTTP/S calls come from same user
- User has to send information of who s/he is at EACH HTTP/S call
- But users can **lie**… (eg, hackers)

Ids and Passwords

- A user will be registered with a *unique* id
- Need also secret password to login
  - Otherwise anyone could login with the ids of other users…
- HTTP/S does not prevent attempts to login to accounts of other users


Log in

Don't have an account? Create one.

Username:

Password:

Remember me (up to 30 days)

How to implement a login mechanism?

- When talking about security and what to implement on the server, think about HTTP/S messages, *not necessarily coming from browsers*.
- Could have endpoint to get *token* from server given userId/password
  - Use such token on each following request as parameter
- GET /login?**userId=x**&**password=y**
  - userId/password as URL parameters to the /login endpoint
  - get back new token Z associated to this user, as HTTP/S response body, no HTML page
- GET /somePageIWantToBrowse**?token=z**
  - pass “token=z” parameter to each HTTP/S request

Awful Solution

- That solution would work, but…
- “*/login?userId=x&password=y*” would be *cached* in your browser history, even after you logout
- How to handle the adding of “*?token=z*” to all your *<a>* tags in the HTML pages?
  - doable, but quite cumbersome
- How to handle browser bookmarks?
  - tokens would be there, and made the links useless once they expire, eg after a logout

POST and Cookies

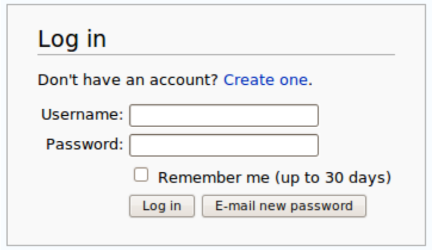
- User ids and passwords should *never* be sent with a GET
  - GET specs do not allow body in the requests
- Should be in HTTP body of a POST
  - This is typical case in HTML forms
- Authentication “tokens” should not be in URLs, but in the HTTP Headers
- **Cookie**: special header that will be used to identify the user
- The user does not choose the cookie, it is the server that assigns them
- Recall: user can craft its own HTTP messages, so server needs to know if cookie values are valid

Login with Cookies

- Browser: POST /login
  - Username X and password as HTTP body
- Server: if login is successful, respond to the POST with a "*Set-Cookie*" header, with some *unique* and *non-predictable* identifier Y
  - Server needs to remember that cookie Y is associated with user X
  - *Set-Cookie: <cookie-name>=<cookie-value>*
- Browser: from now on, each following HTTP request will have "*Cookie:* Y" in the headers
- Logout: remove association between cookie Y and user X on server.
- Server: HTTP request with no cookie or invalid/expired cookie, do 3xx redirect to login page

Request Login page

GET /login.html



Log in

Don't have an account? Create one.

Username:

Password:

Send credentials by submitting the form

POST /login.html
username=foo&password=bar

Validate the credentials. If correct, create a session, identified by a cookie id

HTTP/1.1 302
Set-cookie: **123456**
Location: /index.html

Automatically follow the 302 redirection, and add cookie header in all following requests

GET /index.html
Cookie: **123456**

Cookies and Sessions

- Servers would usually send a "*Set-Cookie*" regardless of login
  - want to know if requests are coming from same user, regardless if s/he is registered/authenticated
  - ie cookies used to define "sessions"
- After login could create a new session (ie, invalidate old cookie and create a new one) or use the existing session cookie (eg, the one set by the server when login page was retrieved with the first GET)
- Problem with re-using session cookies: make sure all the pages were served with HTTPS and not HTTP
  - Ie, use HTTPS for all pages, even the login one
  - do not use HTTP and then switch to HTTPS once login is done

Storing cookies

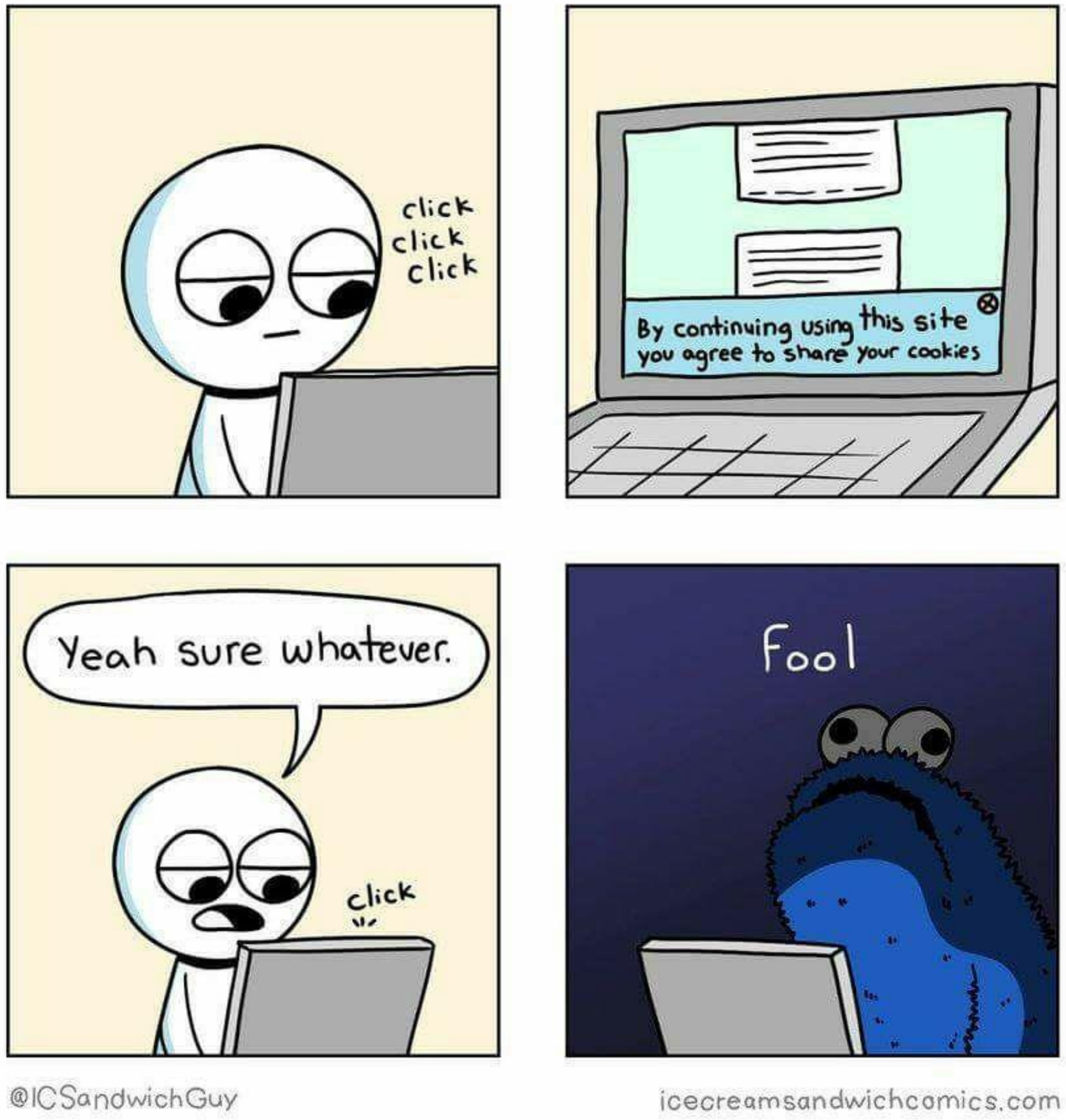
- The browser will store cookie values locally
- At each HTTP/S request, it will send the cookies in the HTTP headers
- Cookies are sent only to same server who asked to set them
  - eg, cookies set from “*foo.com*” are not going to be sent when I do GET requests to “*bar.org*”
- JavaScript can read those cookie values on the browser
- What is the problem with it?
  - You can fabricate a web site with JS that reads all cookies, and send them back to you, so that you can use them to access the user’s Google/Facebook/Bank accounts
- As cookies are arbitrary strings, they can be used to store data
  - usually up to 4K bytes per domain can be stored in a browser

Expires / Secure / HttpOnly

- **Set-Cookie: <name>=<value>; Expires=<date>; Secure; HttpOnly**
- *Expires*: for how long the cookie should be stored
- *Secure*: browser should send the cookie only over HTTPS, and NEVER on HTTP
  - There are kinds of attacks to trick a page to make a HTTP toward the same server instead of HTTPS, and so could read authentication cookies in plain text on the network
- *HttpOnly*: do not allow JS in the browser to read such cookie
  - This is critical for authentication cookies

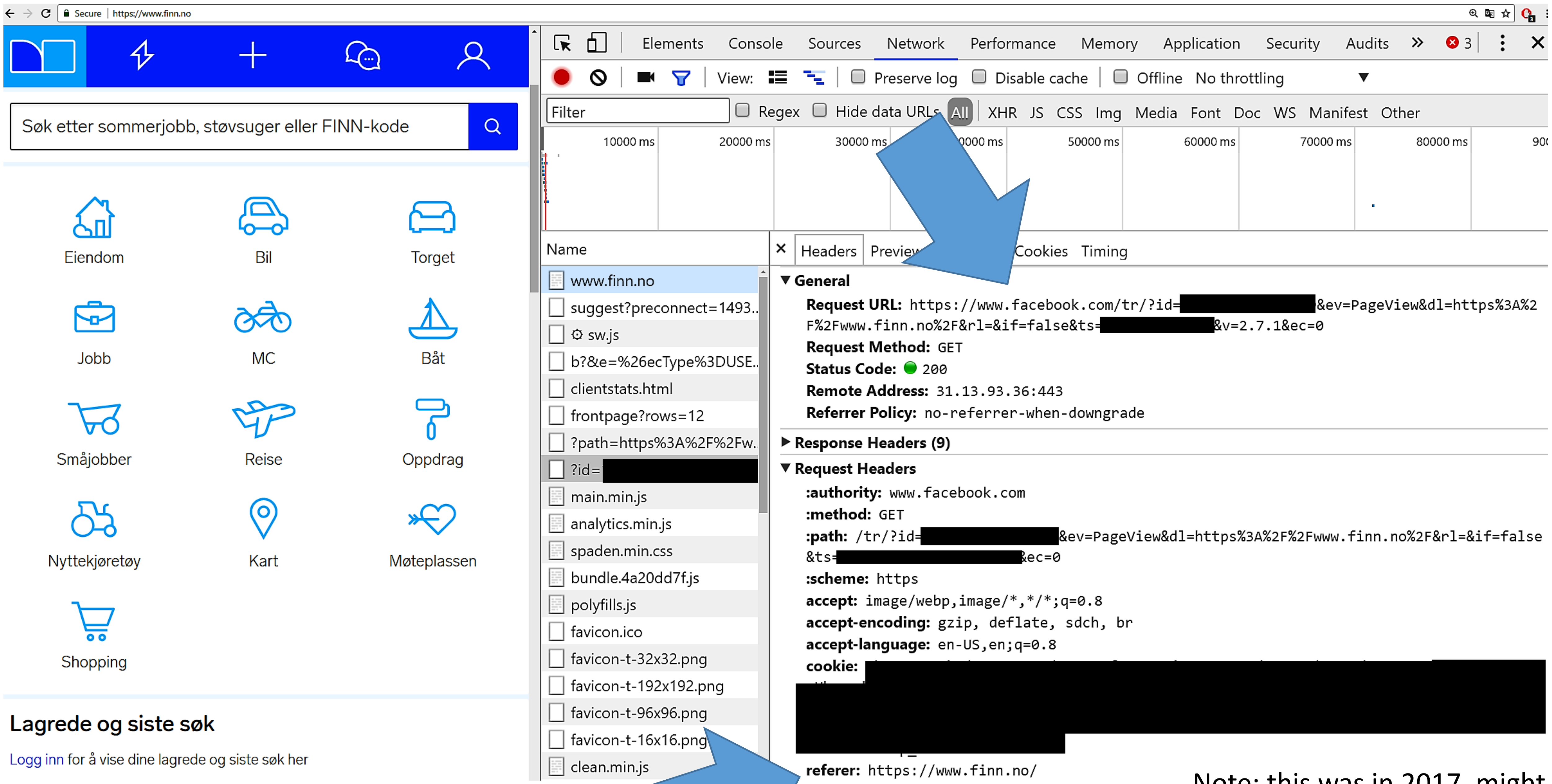
Cookie Tracking

- Besides session/login cookies that have an expiration date, server can setup further cookies (ie Set-Cookie header)
- There are special laws regarding handling of cookies
- Why? Tracking and privacy concerns…


click
click
click
By continuing using this site you agree to share your cookies
Yeah sure whatever.
click
Fool
@ICSandwichGuy
icecreamsandwichcomics.com

Tracking

- Many sites might rely on resources provided by other sites
  - Images, JavaScript files, CSS files, etc.
  - eg, Facebook “Like” button
- When you download a HTML page from domain X (eg *finn.no*) which uses a resource from Y (eg, *facebook.com*), the HTTP GET request for Y will include previous cookies from Y
- So, even if you are logged out from Facebook, FB can know which pages you visit (as long as they do use FB resources), as can have permanent cookies stored on your browser and not related to a current session on FB
- Even worse, FB can track your browser even if you have never used FB!!!
- This happens by simply opening the page from X, no need to click anything!!!
- *referer* HTTP header: domain origin of request to Y from page not from Y
  - Eg, “Referer: X” is added when page loaded from X ask for resource in Y



```
www.finn.no
suggest?preconnect=1493..
sw.js
b?&e=%26ecType%3DUSE..
clientstats.html
frontpage?rows=12
?path=https%3A%2F%2Fw..
?id=[redacted]
main.min.js
analytics.min.js
spaden.min.css
bundle.4a20dd7f.js
polyfills.js
favicon.ico
favicon-t-32x32.png
favicon-t-192x192.png
favicon-t-96x96.png
favicon-t-16x16.png
clean.min.js
```

```
▼ General
  Request URL: https://www.facebook.com/tr/?id=[redacted]&ev=PageView&dl=https%3A%2F%2Fwww.finn.no%2F&rl=&if=false&ts=[redacted]&v=2.7.1&ec=0
  Request Method: GET
  Status Code: ● 200
  Remote Address: 31.13.93.36:443
  Referrer Policy: no-referrer-when-downgrade
▶ Response Headers (9)
▼ Request Headers
  :authority: www.facebook.com
  :method: GET
  :path: /tr/?id=[redacted]&ev=PageView&dl=https%3A%2F%2Fwww.finn.no%2F&rl=&if=false&ts=[redacted]&ec=0
  :scheme: https
  accept: image/webp,image/*,*/*;q=0.8
  accept-encoding: gzip, deflate, sdch, br
  accept-language: en-US,en;q=0.8
  cookie: [redacted]
  referer: https://www.finn.no/
```

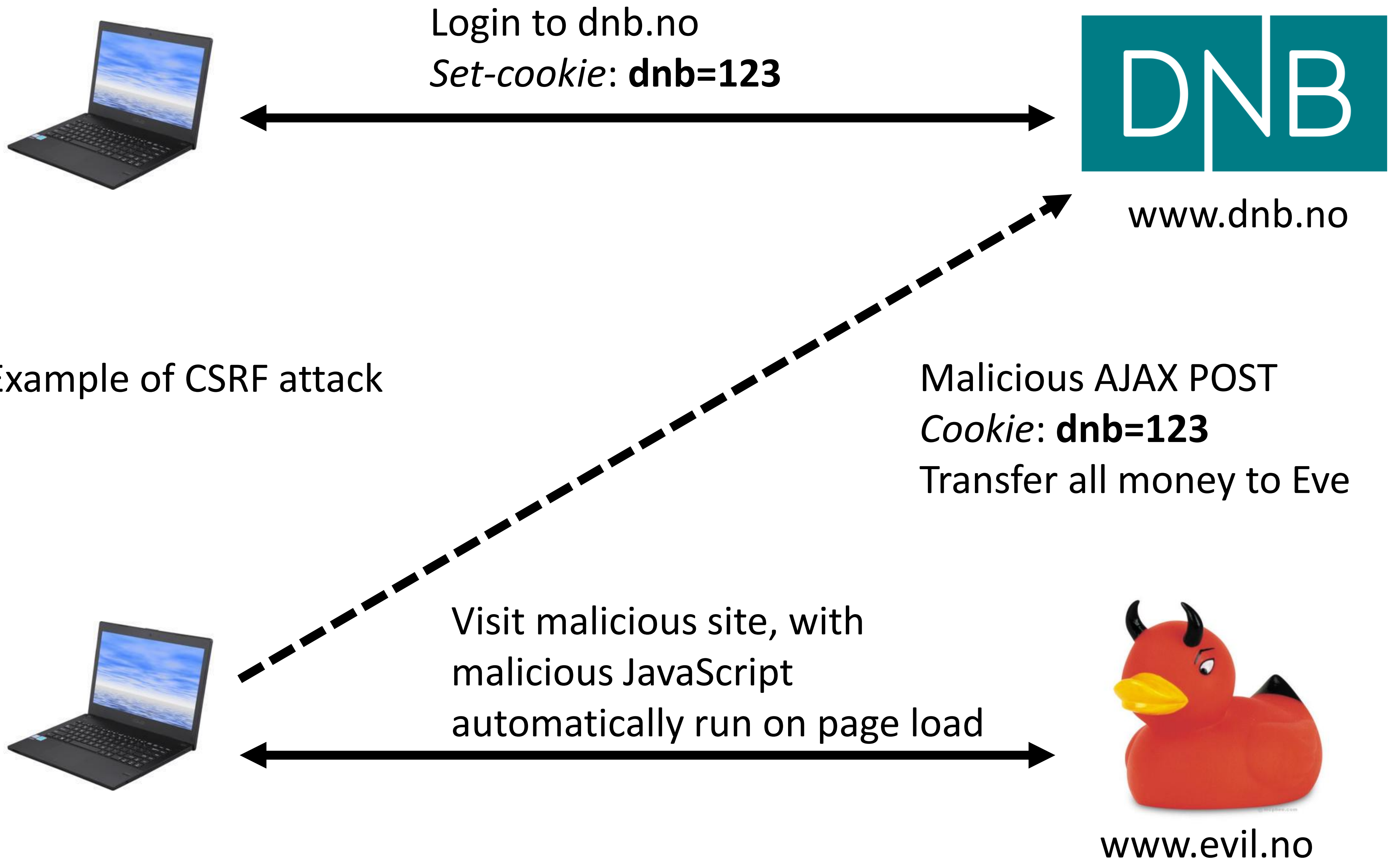
Note: this was in 2017, might have been removed/changed by now on Finn

CSRF and CORS

AJAX and Cookies

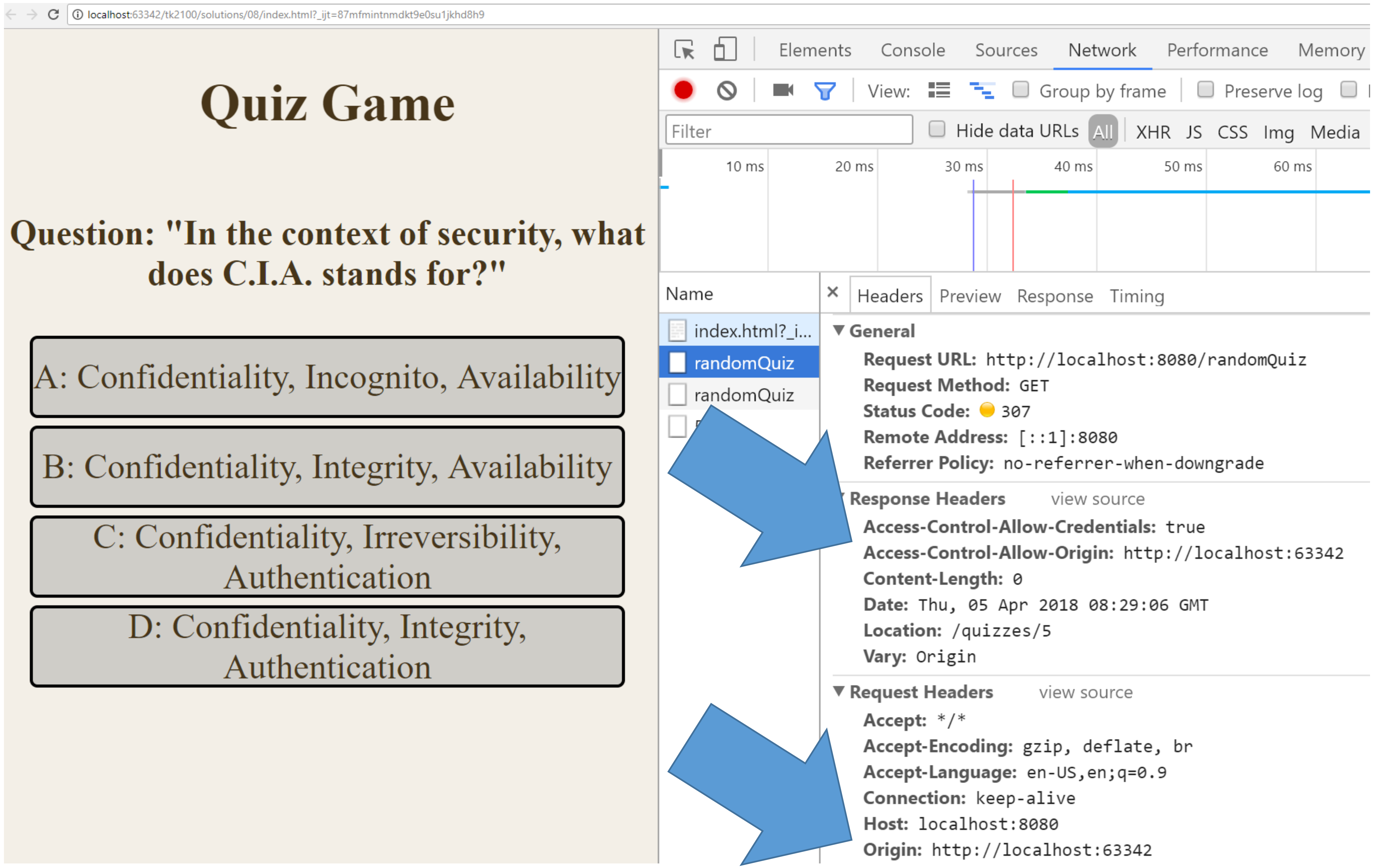
- When browser requests resource for “*foo.com*”, all cookies set by that domain are sent in the headers, session ones included
- This applies also to AJAX requests made with XMLHttpRequest
- *Do you see the problem here?*
- Cross-Site Request Forgery (CSRF) attack

Example of CSRF attack


Login to dnb.no
Set-cookie: **dnb=123**
DNB
www.dnb.no
Malicious AJAX POST
Cookie: **dnb=123**
Transfer all money to Eve
Visit malicious site, with
malicious JavaScript
automatically run on page load
www.evil.no

Cross-Origin Resource Sharing (CORS)

- By default, browsers will allow only AJAX calls toward the same domain (ip:port) of where the JS was downloaded from
  - eg, JS downloaded from *evil.no* can only do AJAX towards *evil.no*
- *Access-Control-Allow-Origin*: special HTTP header set by server to allow requests from other origins/servers
  - "*Access-Control-Allow-Origin: foo.com*" allow only from "*foo.com*"
  - "*Access-Control-Allow-Origin:* *****" allow from anywhere (not really secure at all…)
- *Origin*: special HTTP header, set by browser when making request, specifying origin of the JS



Quiz Game

Question: "In the context of security, what does C.I.A. stands for?"

A: Confidentiality, Incognito, Availability

B: Confidentiality, Integrity, Availability

C: Confidentiality, Irreversibility, Authentication

D: Confidentiality, Integrity, Authentication

General

```
Request URL: http://localhost:8080/randomQuiz
Request Method: GET
Status Code: 307
Remote Address: [::1]:8080
Referrer Policy: no-referrer-when-downgrade
```

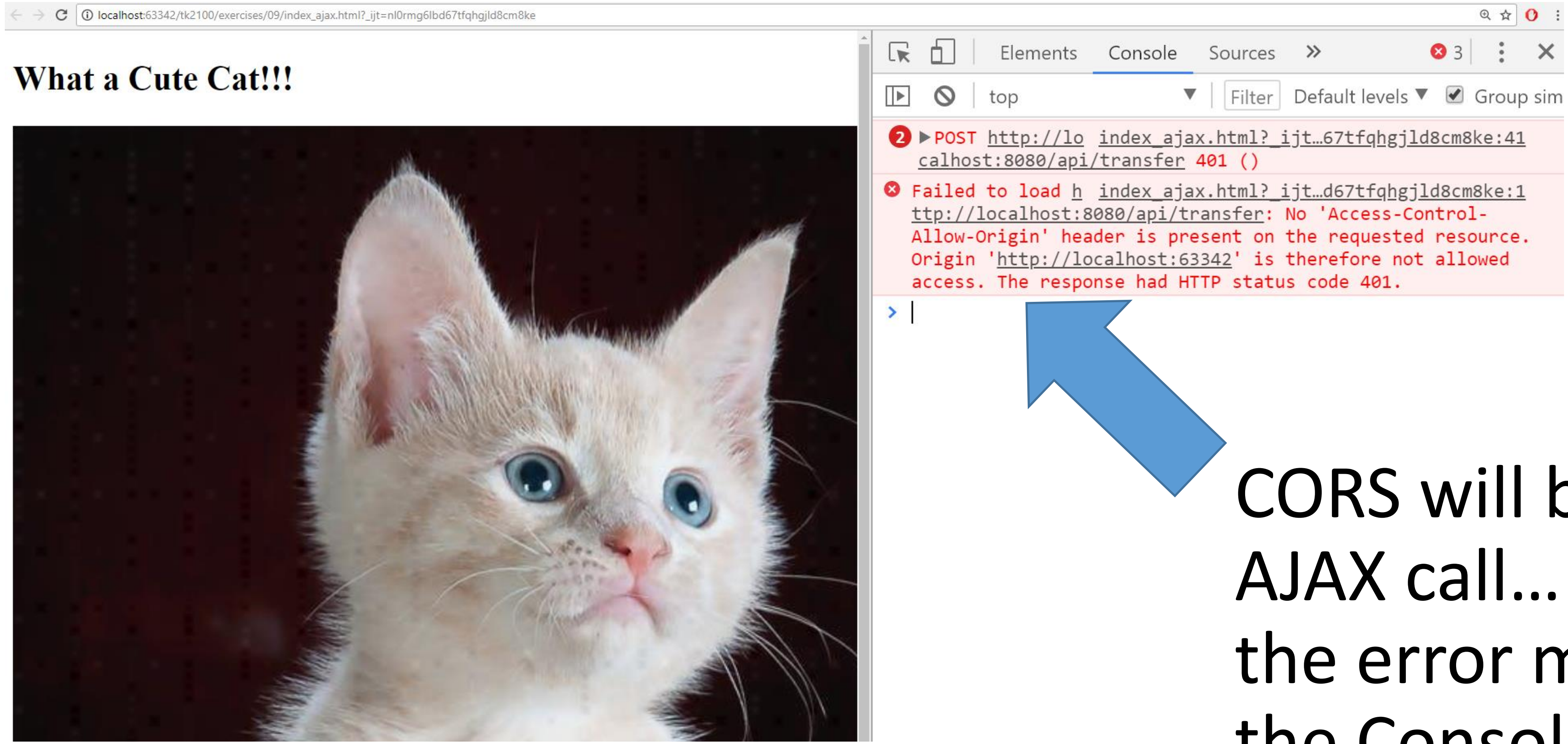
Response Headers

```
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: http://localhost:63342
Content-Length: 0
Date: Thu, 05 Apr 2018 08:29:06 GMT
Location: /quizzes/5
Vary: Origin
```

Request Headers

```
Accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Connection: keep-alive
Host: localhost:8080
Origin: http://localhost:63342
```

Malicious page trying AJAX call toward your bank


localhost:63342/tk2100/exercises/09/index_ajax.html?_ijt=nl0rmg6lbd67tfqhgjld8cm8ke
Elements
Console
Sources
3
top
Filter
Default levels
Group sim

What a Cute Cat!!!

```
2 ▶POST http://lo  index_ajax.html?_ijt…67tfqhgjld8cm8ke:41
calhost:8080/api/transfer 401 ()
Failed to load h  index_ajax.html?_ijt…d67tfqhgjld8cm8ke:1
ttp://localhost:8080/api/transfer: No 'Access-Control-
Allow-Origin' header is present on the requested resource.
Origin 'http://localhost:63342' is therefore not allowed
access. The response had HTTP status code 401.
```

CORS will block the AJAX call… can see the error message in the Console

CORS Not Enough

- CORS can prevent malicious AJAX, but AJAX is not the only way to do HTTP calls in a browser…
- What about if Eve creates page with malicious HTML form toward a bank?

```
<form name=“evilForm"
      action="http://dnb.no/transferMoney"
      method="POST">
    <input name="to" value="eve">
    <input name="amount" value="1000">
</form>
```

Two Problems for Eve

1. How to trick the user to **click** on the form to submit it?
2. How to **hide** the fact that there is such malicious form in the HTML page so that the user has no idea of what is going on?

Can submit forms with JS…

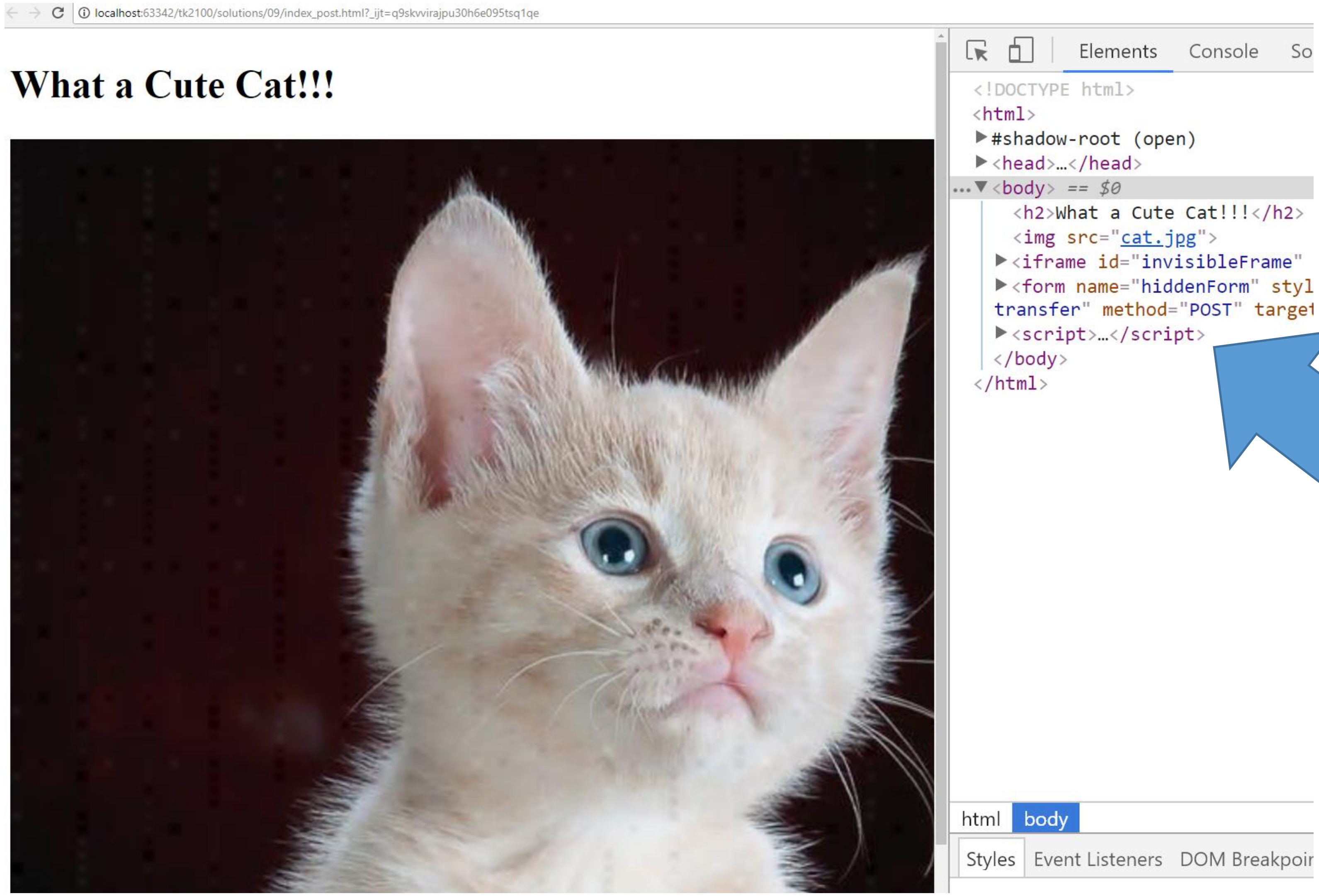
document.forms[“evilForm”].submit();

Can execute it once page is loaded… d’oh!!!

CSS Hiding

- *“display:none”*
  - can use CSS to hide the presence of HTML elements in the page…
- *<iframe id="invisibleFrame" name="invisibleFrame" style="display:none"></iframe>*
- *<form … target="invisibleFrame“>*
  - can redirect output/result of the HTTP POST to an invisible frame

Example of malicious page with hidden form



localhost:63342/tk2100/solutions/09/index_post.html?_ijt=q9skvvirajpu30h6e095tsq1qe

What a Cute Cat!!!

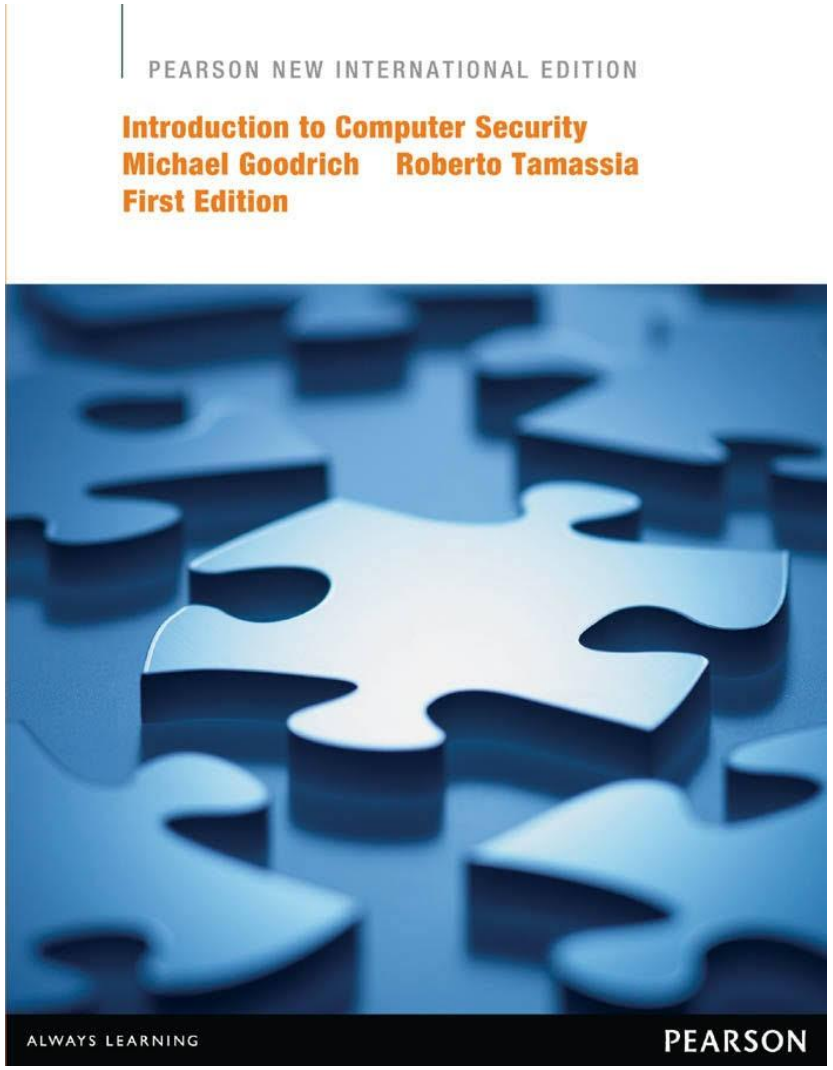
```
<!DOCTYPE html>
<html>
 ▶#shadow-root (open)
 ▶<head>…</head>
…▼<body> == $0
    <h2>What a Cute Cat!!!</h2>
    <img src="cat.jpg">
   ▶<iframe id="invisibleFrame"
   ▶<form name="hiddenForm" styl
  transfer" method="POST" target
   ▶<script>…</script>
  </body>
</html>
```

Automatically submit the hidden form with JavaScript.. CORS is not going to help you here…

CSRF Prevention

- Using a cookie token for authentication is not enough for critical operations
- Need a *second token* which is **NOT** stored in a cookie, and added in the HTML/JS at each request
- Server will validate both tokens before accepting a request
- Details on how to handle such token depends on the framework used by the server, and how it generates the HTML pages
  - eg, added as a secret, non-visible input in the HTML forms
- *Eve will/should not be able to read and use such second token*

For Next Week



PEARSON NEW INTERNATIONAL EDITION

Introduction to Computer Security
Michael Goodrich Roberto Tamassia
First Edition

ALWAYS LEARNING PEARSON

- Book pages: 334-338, 342-348, 356, 364-367
- Note: when I tell you to **study** some specific pages in the book, it would be good if you also *read* the other pages in the same chapter at least once
- Exercises for Lesson 9 on GitHub repository