Software Security
Sessions, Cookies and Threats

Alessandra Gorla

institute imdea software

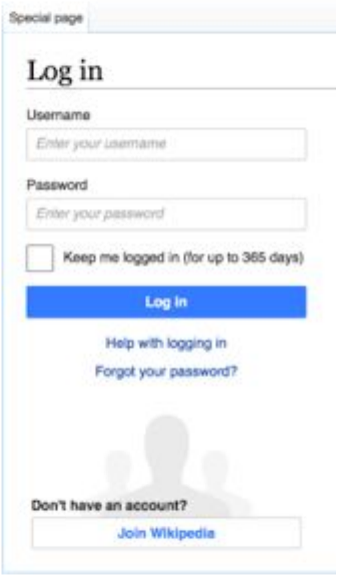
Goals for today

- Authentication in Web apps
- Sessions and cookies
- Threats on using cookies
    - Session hijacking
    - Cookie forging and poisoning
    - Cross-Site Request Forgery
        - Cross-origin Resource Sharing
        - other mitigations

Authentication on Web apps

IDs and Passwords

- A user will be registered with a unique id
- Need also secret password to login
  - HTTP/S does not prevent attempts to login to accounts of other users



HTTP is stateless

- Server does not know who the user is
- Server only sees incoming HTTP/S messages
- 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…

How to implement a login mechanism?

- Client gets 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 browser bookmarks?
  - tokens would be there, and would make links useless once they expire, eg after a logout

Cookies

POST and cookies

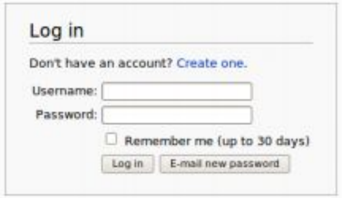
- User ids and passwords should **never** be sent with a GET
  - GET specs do not allow body in the requests
- Should be in HTTP body of a POST
  - This is typical case in HTML forms
- Authentication “tokens” should not be in URLs, but in the HTTP Headers
- **Cookie**: special header that will be used 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:

□ Remember me (up to 30 days)

Log in

E-mail new 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 or use the existing session cookie
- 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

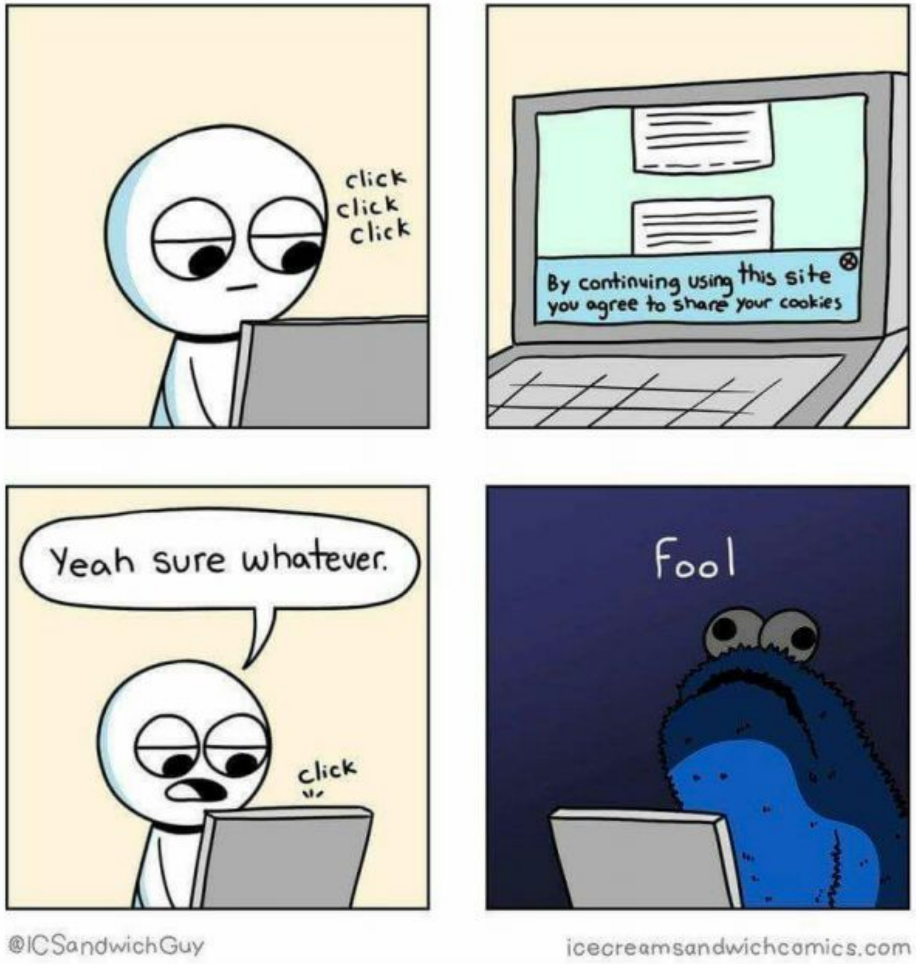
- The browser will store cookie values locally
- At each HTTP/S request, it will send the cookies in the HTTP headers
- Cookies are sent only to same server who asked to set them
    - eg, cookies set from “foo.com” are not going to be sent when I do GET requests to “bar.org”
- JavaScript can read those cookie values on the browser
- As cookies are arbitrary strings, they can be used to store data

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

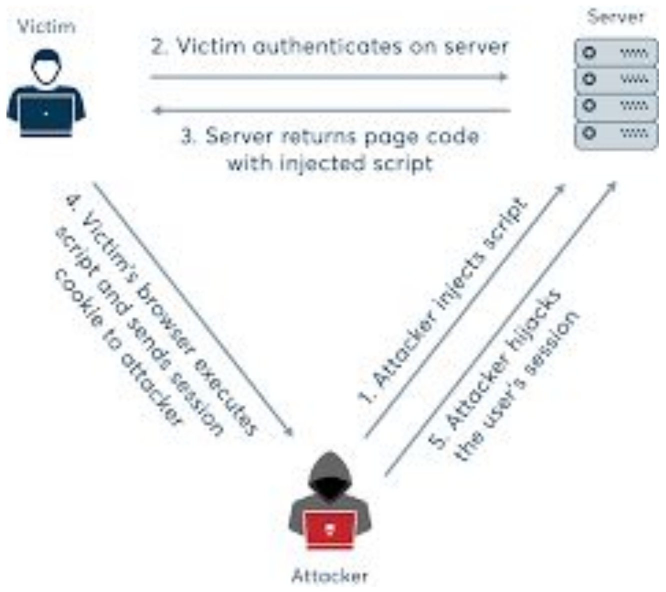
Tracking

- 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 elpais.es) which uses a resource from Y (eg, facebook.com), the HTTP GET request for Y will include previous cookies from Y

Session Hijacking

Session hijacking

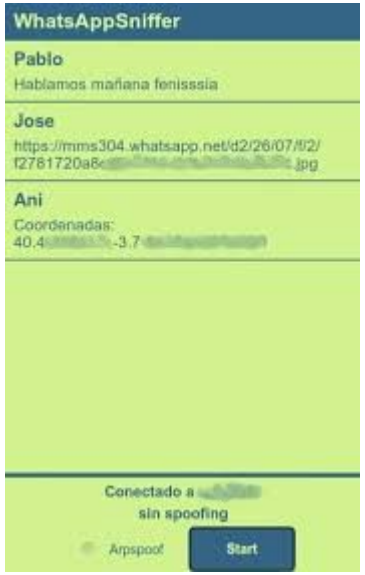
- Adversaries with access to a session cookie can perform session hijacking, allowing them to log in as the user and access *any resources that user can access.*


Victim
Server
2. Victim authenticates on server
3. Server returns page code with injected script
4. Victim's browser executes script and sends session cookie to attacker
1. Attacker injects script
5. Attacker hijacks the user's session
Attacker

Session side-jacking

- A web application might be vulnerable to session side-jacking if it uses SSL for login pages but not for the rest of the site after a user has authenticated
- An attacker can read packets sent after authentication—including the plaintext session cookie—and can generate packets with the same session cookie.
- Those packets will be interpreted by the server as packets coming from an authenticated user, allowing the attacker to access any resources the victim had permissions for.

Session side-jacking



Pablo

Hablamos mañana fenisssia

Jose

https://mms304.whatsapp.net/d2/26/07/f/2/
f2781720a8e[redacted].jpg

Ani

Coordenadas:
40.4[redacted] -3.7[redacted]

Conectado a [redacted]
sin spoofing

○ Arpspoof

Mitigating session side-jacking

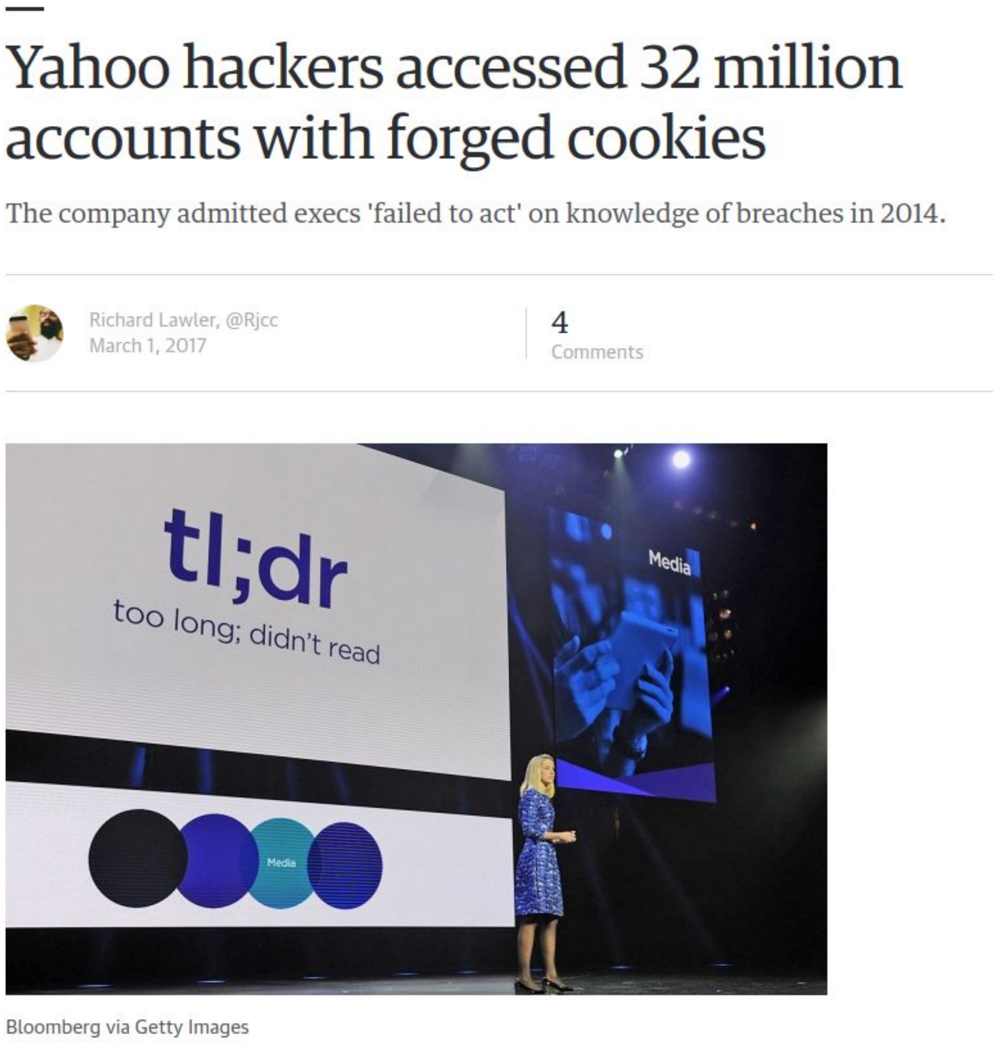
- The most common strategy for mitigating session side-jacking is to always run HTTP over a secure (SSL) channel

Cookie forging

Cookie forging

- If session cookies are predictable, an attacker can simply guess the session cookie and send a request with that cookie value

Yahoo in 2017



Yahoo hackers accessed 32 million accounts with forged cookies

The company admitted execs 'failed to act' on knowledge of breaches in 2014.

Richard Lawler, @Rjcc
March 1, 2017

4
Comments

Bloomberg via Getty Images

Mitigating cookies forging

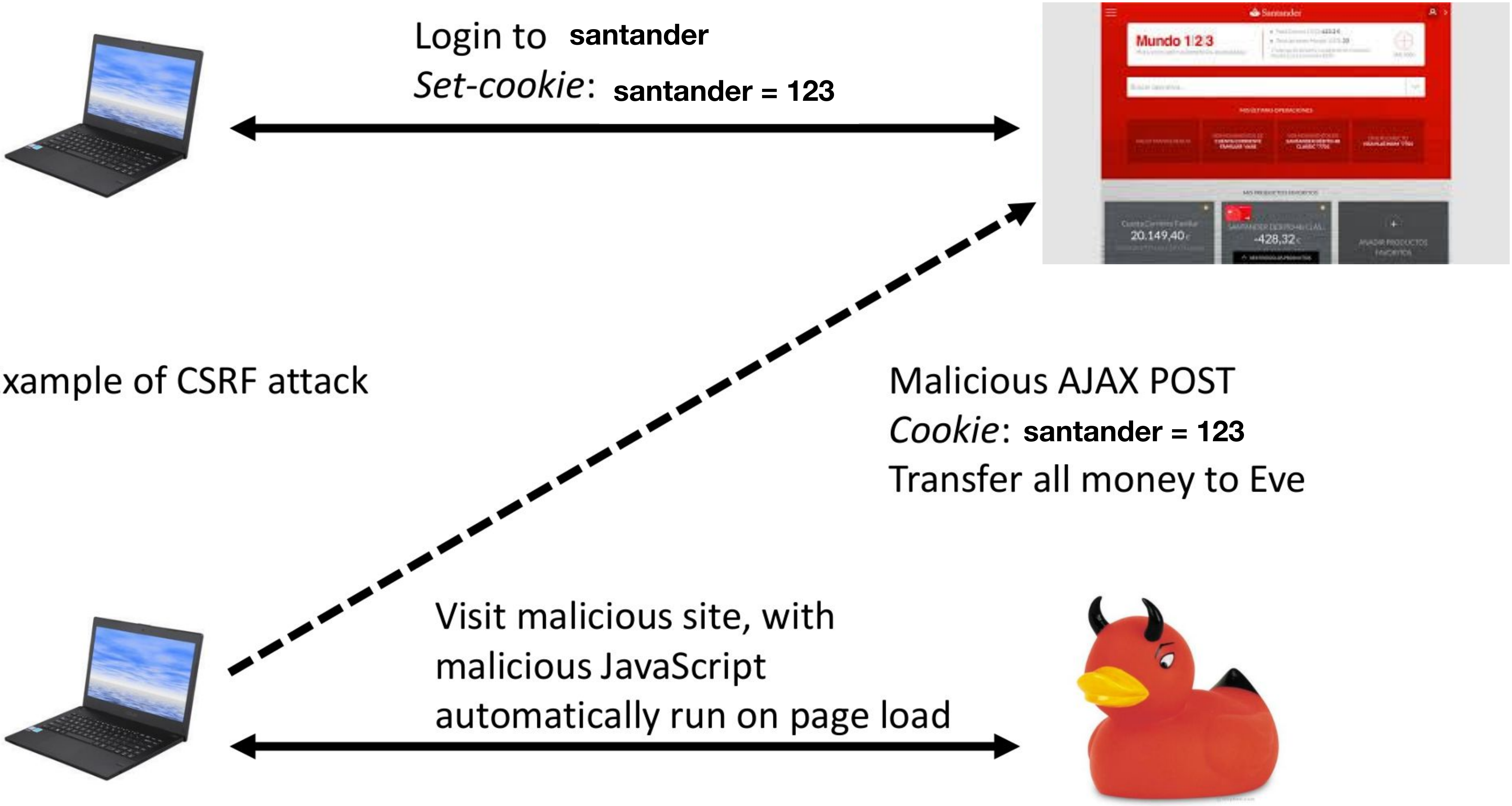
- Calls for using long, random numbers or strings as session ids to minimize the probability of successful cookie forging.

Cross-Site Request Forgery

Cross-Site Request Forgery (CSRF)

- Many users remain logged in to accounts (e.g., Gmail, Facebook) even when they are not actively using a site.
- This means that their browser has an active session cookie for that site stored in its local state.
- If an attacker can force the target to issue a request to such a site, that request will be sent with a valid session cookie and will be treated as an authenticated request by the user.

CSRF - example



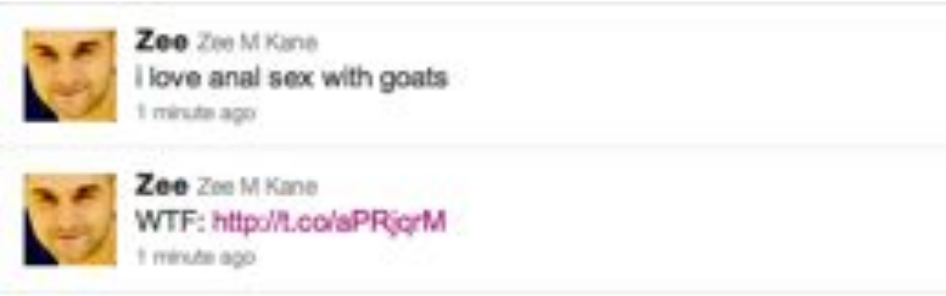
Login to **santander**
Set-cookie: **santander = 123**

Example of CSRF attack

Malicious AJAX POST
Cookie: **santander = 123**
Transfer all money to Eve

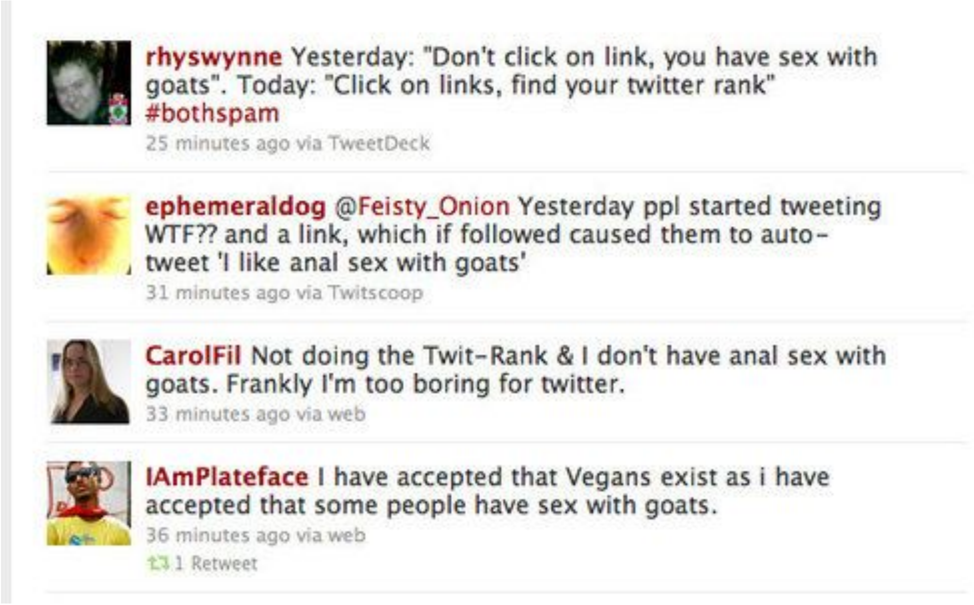
Visit malicious site, with
malicious JavaScript
automatically run on page load

CSRF - Twitter true story...



Zee Zee M Kane
I love anal sex with goats
1 minute ago

Zee Zee M Kane
WTF: http://t.co/aPRjqrM
1 minute ago



rhyswynne Yesterday: "Don't click on link, you have sex with goats". Today: "Click on links, find your twitter rank" #bothspam
25 minutes ago via TweetDeck

ephemeraldog @Feisty_Onion Yesterday ppl started tweeting WTF?? and a link, which if followed caused them to auto-tweet 'I like anal sex with goats'
31 minutes ago via Twitscoop

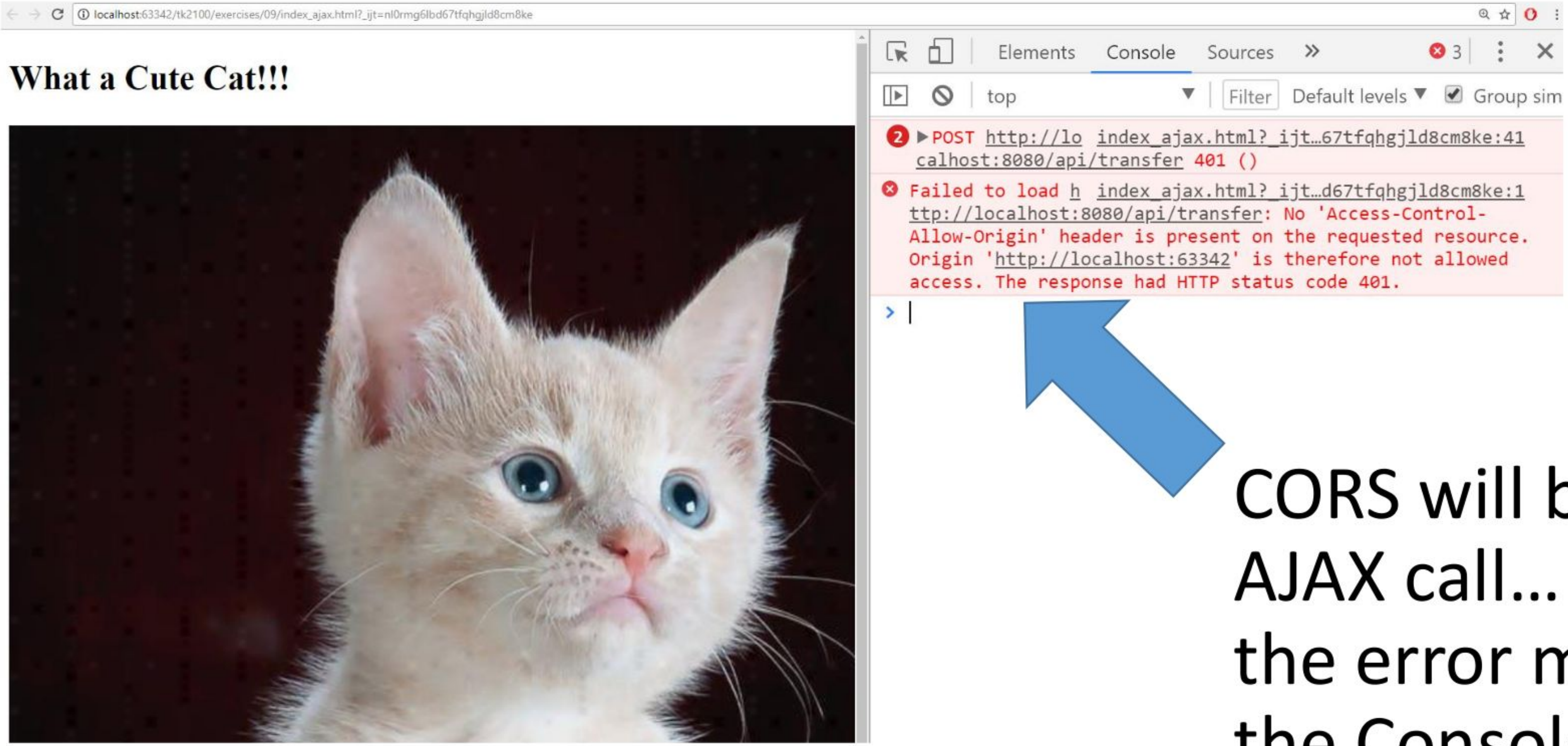
CarolFil Not doing the Twit-Rank & I don't have anal sex with goats. Frankly I'm too boring for twitter.
33 minutes ago via web

IAmPlateface I have accepted that Vegans exist as i have accepted that some people have sex with goats.
36 minutes ago via web
1 Retweet

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.com can only do AJAX towards evil.com
- *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

CORS as mitigation


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 as mitigation

- 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://santander.es/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?

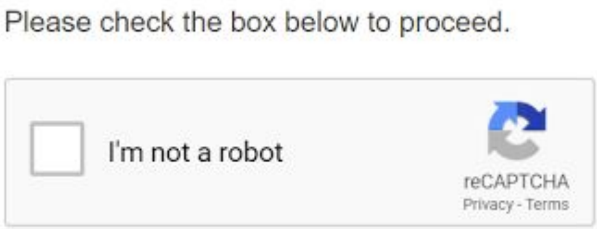
- Can submit forms with JavaScript
    - document.forms[“evilForm”].submit();
- Use CSS to hide presence of HTML elements in the page
    - “display:none”

Mitigating CSRF

- The primary defense against CSRF attacks is for the (target) server to attempt to distinguish between genuine requests and forged requests.
- Techniques for achieving this include
  - secret validation tokens
  - referer validation, and
  - custom HTTP headers .

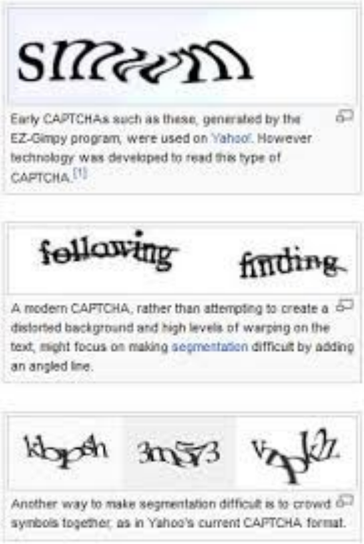
Mitigating CSRF

- CSRF attacks can also be mitigated by requiring user actions (e.g., successful CAPTCHA completion) to authorize requests with side effects.



Please check the box below to proceed.

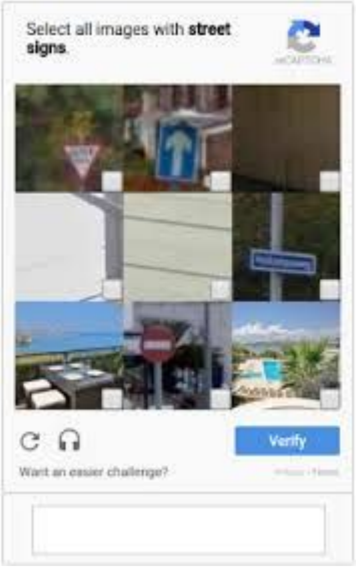
□ I'm not a robot



Early CAPTCHAs such as these, generated by the EZ-Gimpy program, were used on Yahoo!. However technology was developed to read this type of CAPTCHA.[1]

A modern CAPTCHA, rather than attempting to create a distorted background and high levels of warping on the text, might focus on making segmentation difficult by adding an angled line.

Another way to make segmentation difficult is to crowd symbols together, as in Yahoo's current CAPTCHA format.



Select all images with **street signs**.

Want an easier challenge?

Beware! Malware

- Malware that runs on client machine might have access to stored browser state (including session cookies)
- Solution: Chrome browser encrypts local state
- HOWEVER
    - it relies on account-based keys

Beware! Cross-Site scripting

- One type of data that is often the target of XSS attacks is the cookie storing the session id
- Solution: protect against XSS..