Web Security

CS 161: Computer Security

Prof. Raluca Ada Popa

March 19, 2019

The web architecture is a mess when it comes to security

Some content adapted from materials by David Wagner or Dan Boneh

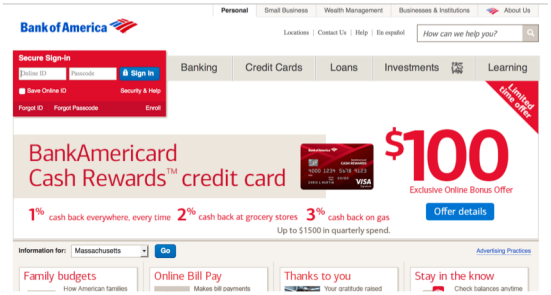
Announcements

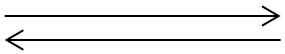
- Homework 3 due Friday,
- Project 1 and 2 grades released

What is the Web?

A platform for deploying applications and sharing information, *portably* and *securely*

client browser





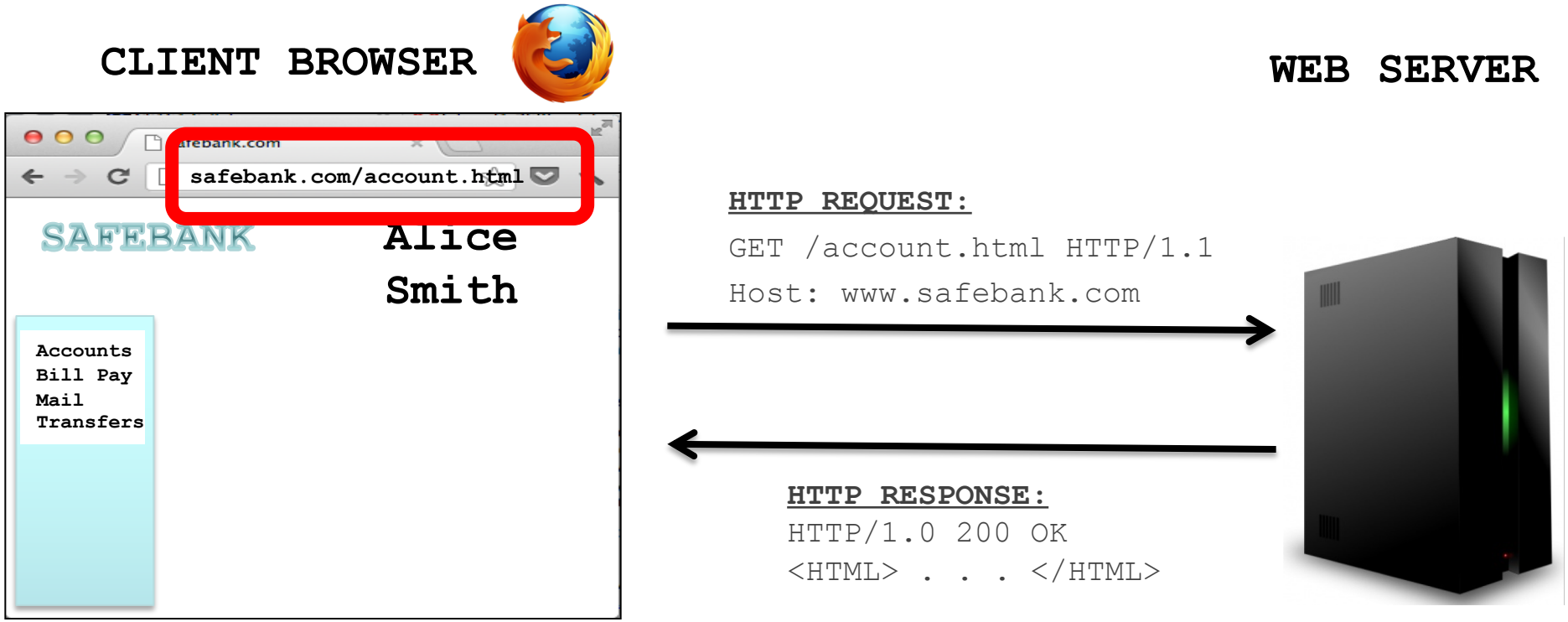
web server

Bank of America

HTTP

(Hypertext Transfer Protocol)

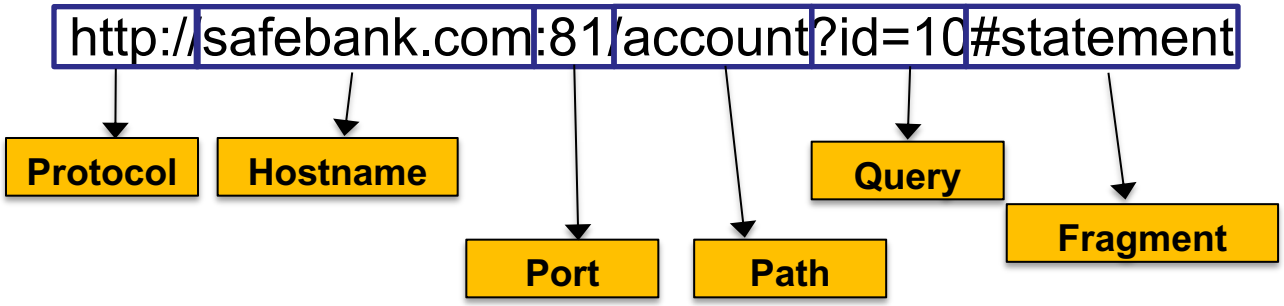
A common data communication protocol on the web



URLs

Global identifiers of network-retrievable resources

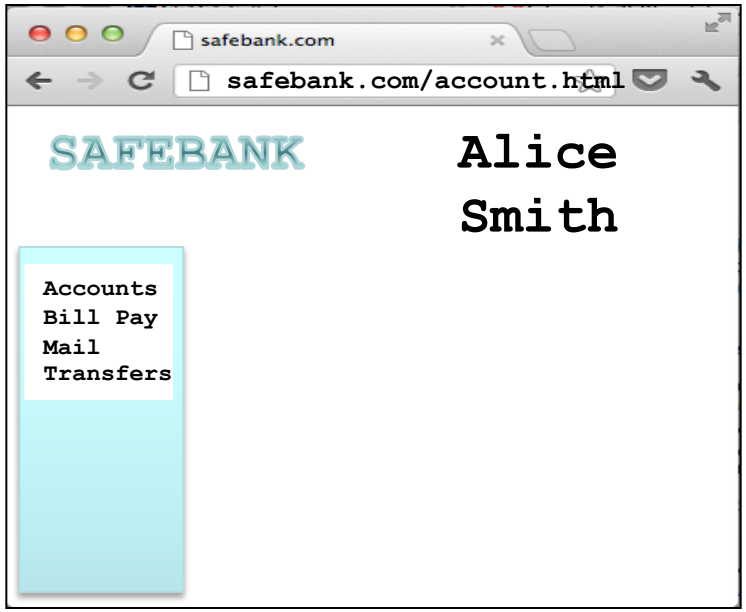
Example:



HTTP

CLIENT BROWSER

WEB SERVER



HTTP REQUEST:

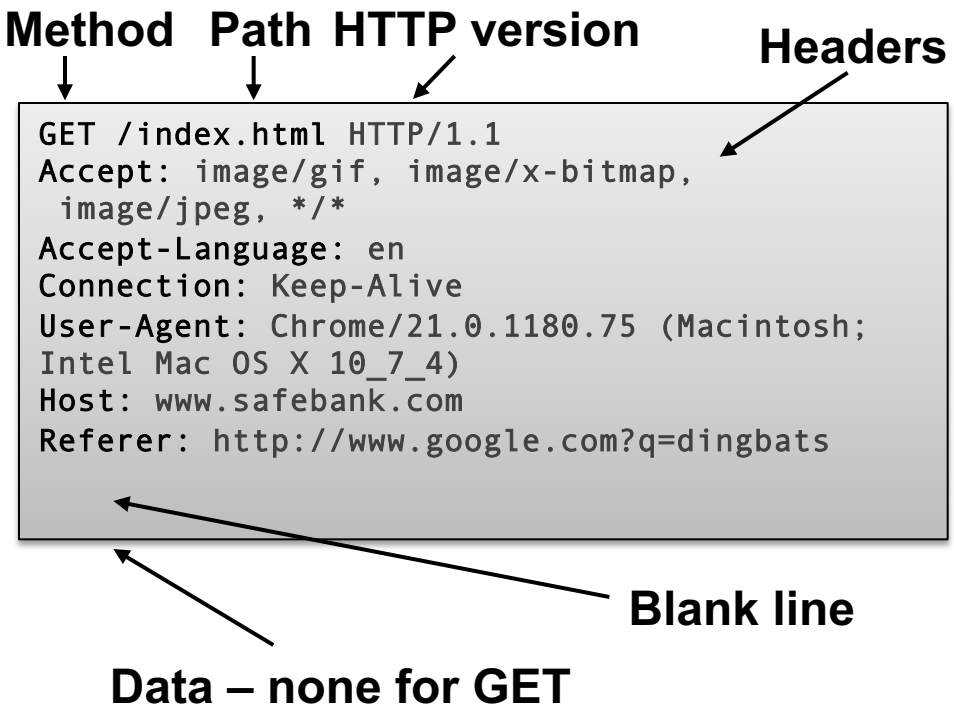
```
GET /account.html HTTP/1.1
Host: www.safebank.com
```

HTTP RESPONSE:

```
HTTP/1.0 200 OK
<HTML> . . . </HTML>
```

HTTP Request

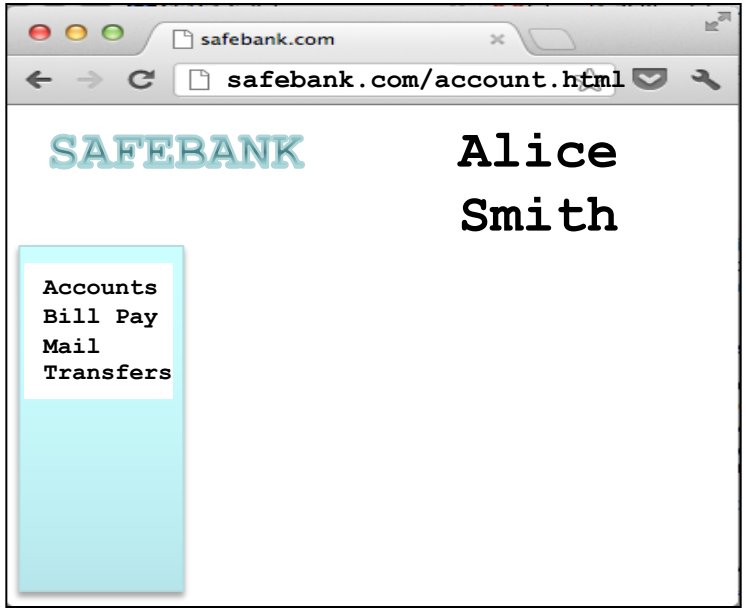
GET: no side effect
POST: possible side effect



HTTP

CLIENT BROWSER

WEB SERVER



HTTP REQUEST:

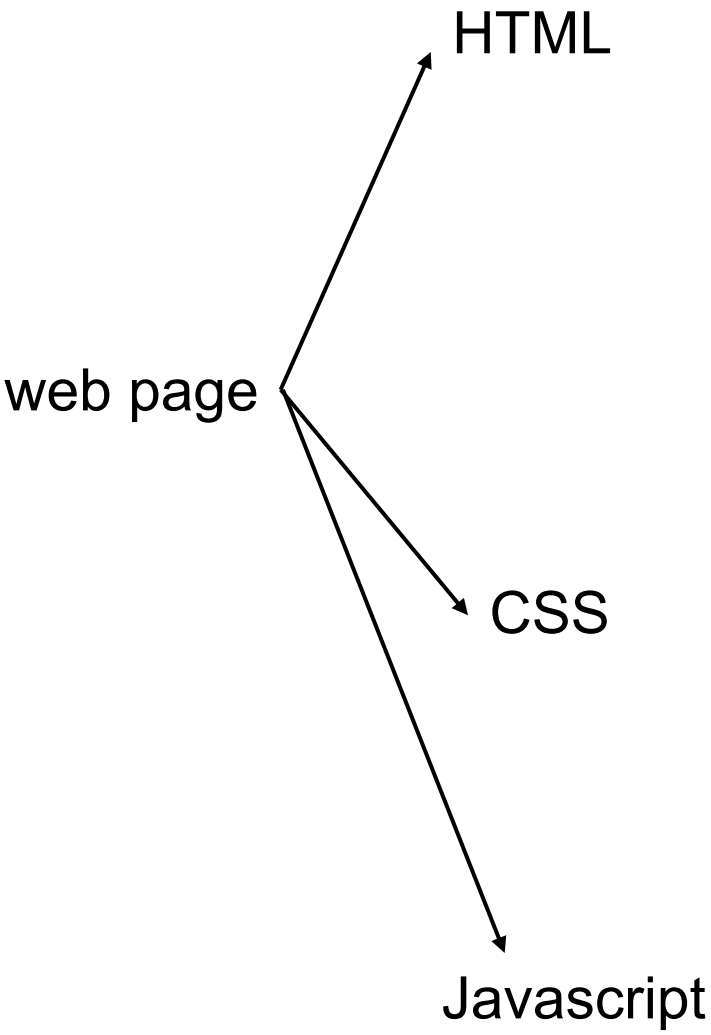
```
GET /account.html HTTP/1.1
Host: www.safebank.com
```

HTTP RESPONSE:

```
HTTP/1.0 200 OK
<HTML> . . . </HTML>
```

HTTP Response

Web page



HTML

A language to create structured documents
One can embed images, objects, or create interactive forms

index.html

```
<html>
    <body>
        <div>
            foo
            <a href="http://google.com">Go to Google!</a>
        </div>
        <form>
            <input type="text" />
            <input type="radio" />
            <input type="checkbox" />
        </form>
    </body>
</html>
```

CSS (Cascading Style Sheets)

Style sheet language used for describing the presentation of a document

index.css

```
p.serif {
font-family: "Times New Roman", Times, serif;
}
p.sansserif {
font-family: Arial, Helvetica, sans-serif;
}
```

Javascript

Programming language used to manipulate web pages. It is a high-level, untyped and interpreted language with support for objects.

Supported by all web browsers

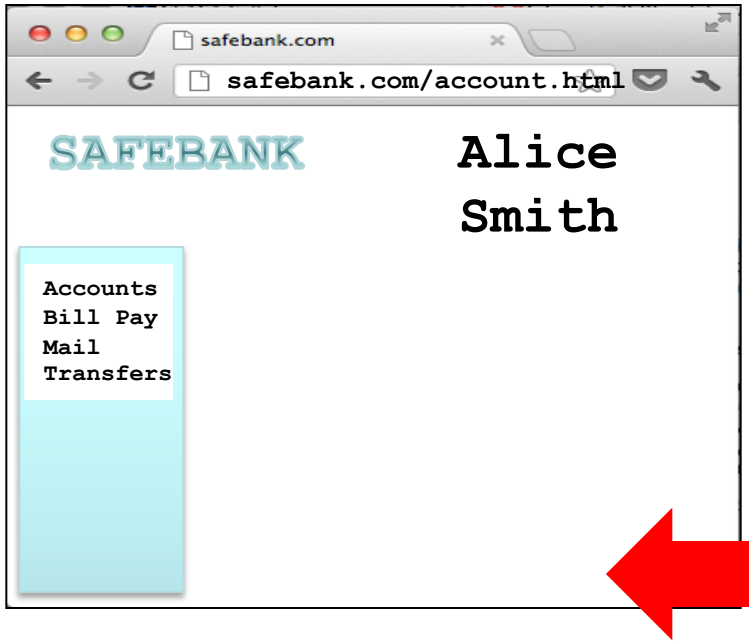
```
<script>
function myFunction() {
document.getElementById("demo").innerHTML = ”Text changed.";
}
</script>
```

Very powerful!

HTTP

CLIENT BROWSER

WEB SERVER



HTTP REQUEST:

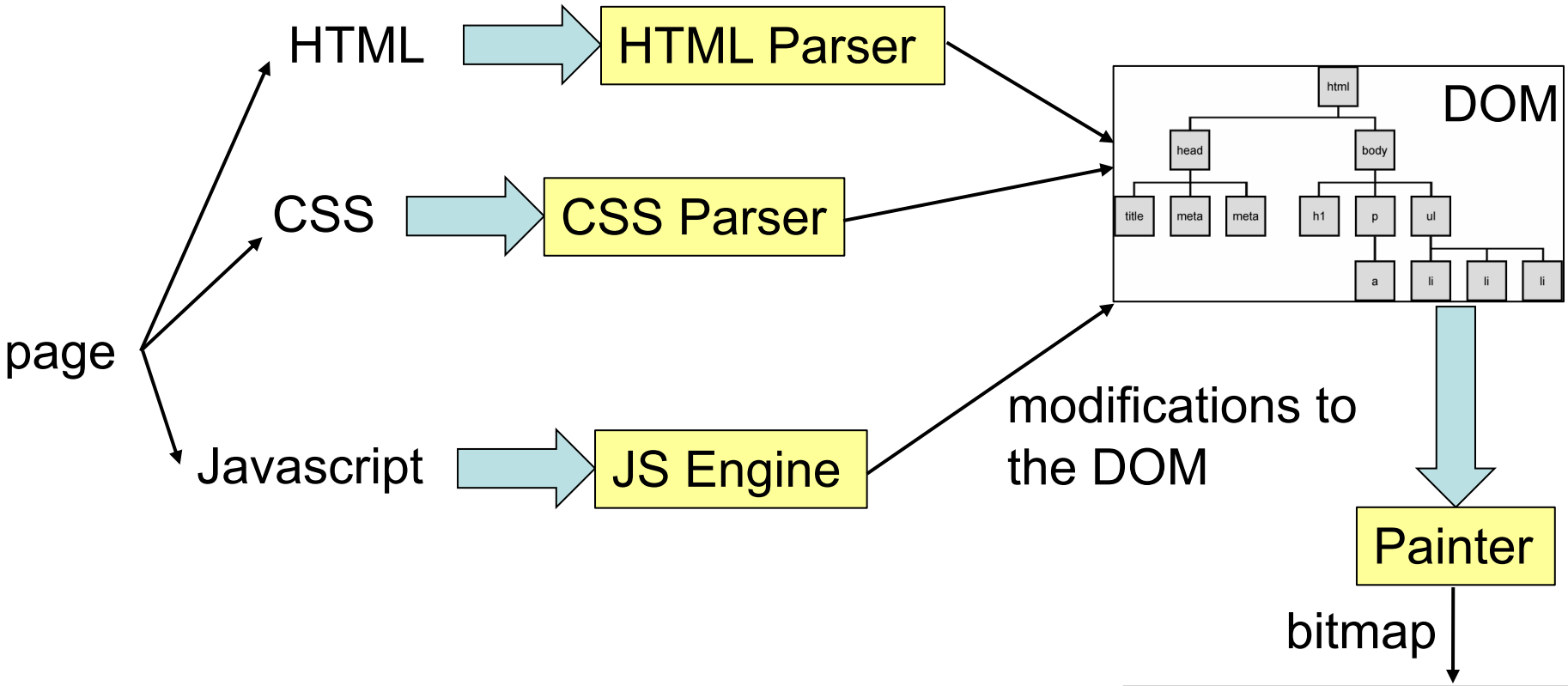
```
GET /account.html HTTP/1.1
Host: www.safebank.com
```

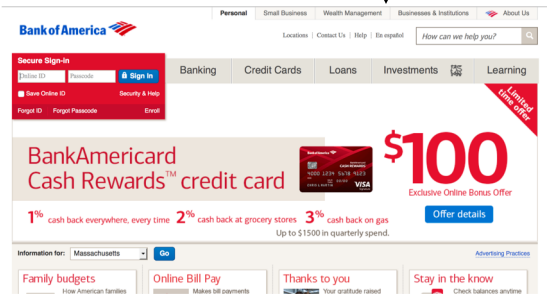
HTTP RESPONSE:

```
HTTP/1.0 200 OK
<HTML> . . . </HTML>
```

webpage

Page rendering



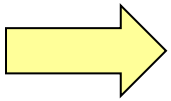


DOM (Document Object Model)

a cross-platform model for representing and interacting with objects in HTML

HTML

```
<html>
    <body>
        <div>
            foo
        </div>
        <form>
            <input type="text" />
            <input type="radio" />
            <input type="checkbox" />
        </form>
    </body>
</html>
```



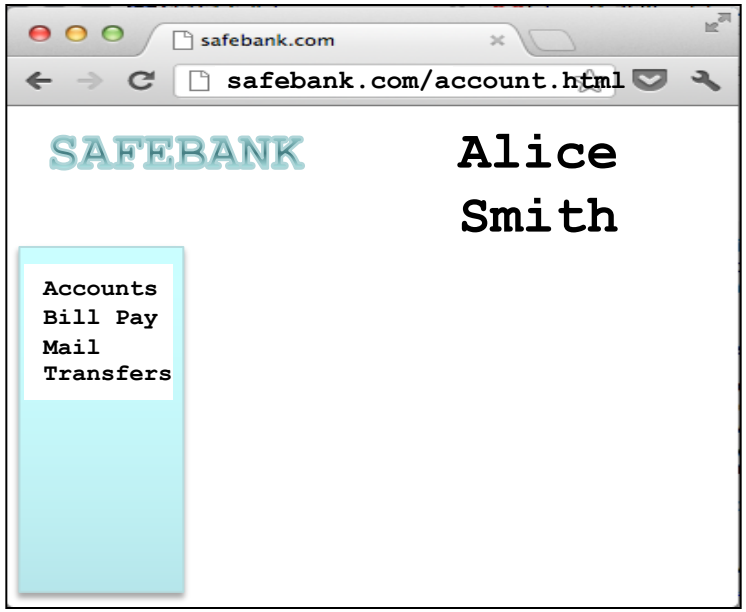
DOM Tree

```
|-> Document
   |-> Element (<html>)
      |-> Element (<body>)
         |-> Element (<div>)
            |-> text node
         |-> Form
               |-> Text-box
               |-> Radio Button
               |-> Check Box
```

Web & HTTP 101

CLIENT BROWSER

WEB SERVER



HTTP REQUEST:

```
GET /account.html HTTP/1.1
Host: www.safebank.com
```

HTTP RESPONSE:

```
HTTP/1.0 200 OK
<HTML> . . . </HTML>
```

The power of Javascript

Get familiarized with it so that you can think of all the attacks one can do with it

What can you do with Javascript?

Almost anything you want to the DOM!

A JS script embedded on a page can modify in almost arbitrary ways the DOM of the page. The same happens if an attacker manages to get you load a script into your page.

w3schools.com has nice interactive tutorials: https://www.w3schools.com/w3css/tryit.asp

Example of what Javascript can do…

Can change HTML content:

```
<p id="demo">JavaScript can change HTML content.</p>

<button type="button"
onclick="document.getElementById('demo').innerHTML =
'Hello JavaScript!'">
   Click Me!</button>
```

DEMO from w3schools.com

Other examples

Can change images
Can chance style of elements
Can hide elements
Can unhide elements
Can change cursor

Other example: can access cookies

Will learn later that cookies are useful for authentication.

JS can read cookie:

```
var x = document.cookie;
```

Change cookie with JS:

```
document.cookie = "username=John Smith; expires=Thu,
18 Dec 2013 12:00:00 UTC; path=/";
```

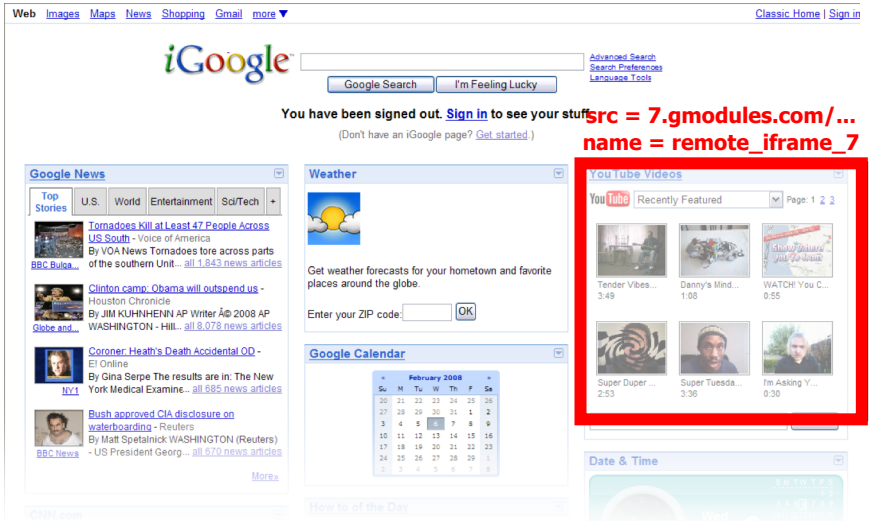
Frames

Frames

- Enable embedding a page within a page

```
<iframe src="URL"></iframe>
```

Frames



- Modularity
  - Brings together content from multiple sources
  - Client-side aggregation
- Delegation
  - Frame can draw only on its own rectangle

Slide from Dan Boneh

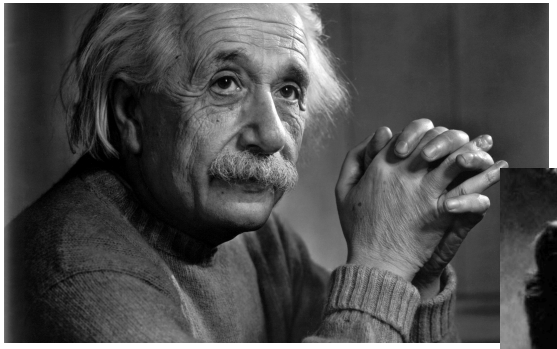
Frames

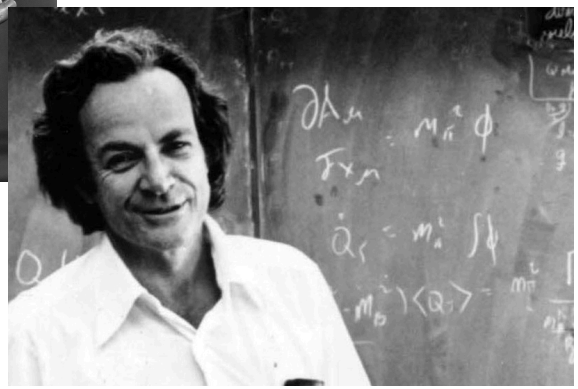
- Outer page can specify only sizing and placement of the frame in the outer page
  - demo
- Frame isolation: Our page cannot change contents of inner page, inner page cannot change contents of outer page

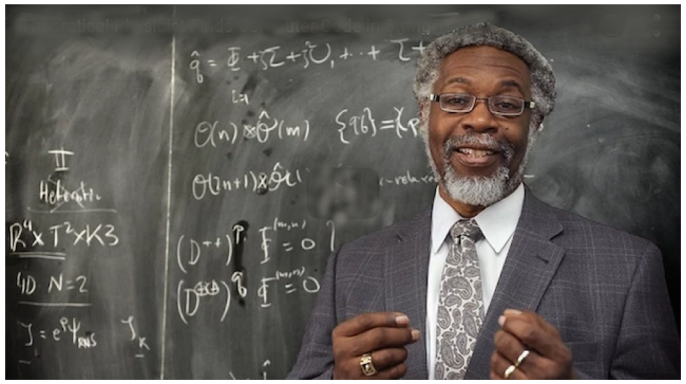
Web security

A historical perspective

- The web is an example of “bolt-on security”
- Originally, the web was invented to allow physicists to share their research papers
  - Only textual web pages + links to other pages; no security model to speak of







The web became complex and adversarial quickly

- Then we added embedded images
  - Crucial decision: a page can embed images loaded from another web server
- Then, Javascript, dynamic HTML, AJAX, CSS, frames, audio, video, …
- Today, a web site is a distributed application
- Attackers have various motivations

Web security is a challenge!

Desirable security goals

- **Integrity:** malicious web sites should not be able to tamper with integrity of my computer or my information on other web sites
- **Confidentiality:** malicious web sites should not be able to learn confidential information from my computer or other web sites
- **Privacy:** malicious web sites should not be able to spy on me or my activities online
- **Availability**: attacker cannot make site unavailable

Security on the web

- Risk #1: we don't want a malicious site to be able to trash my files/programs on my computer
  - Browsing to `awesomevids.com` (or `evil.com`) should not infect my computer with malware, read or write files on my computer, etc.

Security on the web

- Risk #1: we don't want a malicious site to be able to trash my files/programs on my computer
  - Browsing to `awesomevids.com` (or `evil.com`) should not infect my computer with malware, read or write files on my computer, etc.
- Defense: Javascript is sandboxed;
  try to avoid security bugs in browser code;
  privilege separation; automatic updates; etc.

Security on the web

- Risk #2: we don't want a malicious site to be able to spy on or tamper with my information or interactions with other websites
  - Browsing to `evil.com` should not let `evil.com` spy on my emails in Gmail or buy stuff with my Amazon account

Security on the web

- Risk #2: we don't want a malicious site to be able to spy on or tamper with my information or interactions with other websites
  - Browsing to `evil.com` should not let `evil.com` spy on my emails in Gmail or buy stuff with my Amazon account
- Defense: the same-origin policy
  - A security policy grafted on after-the-fact, and enforced by web browsers

Security on the web

- Risk #3: we want data stored on a web server to be protected from unauthorized access

Security on the web

- Risk #3: we want data stored on a web server to be protected from unauthorized access
- Defense: server-side security

Same-origin policy

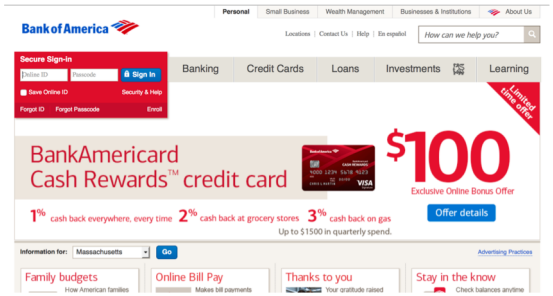
Same-origin policy

- Each site in the browser is isolated from all others

browser:

security barrier

wikipedia.org



mozilla.org

Same-origin policy

- Multiple pages from the same site are not isolated

browser:

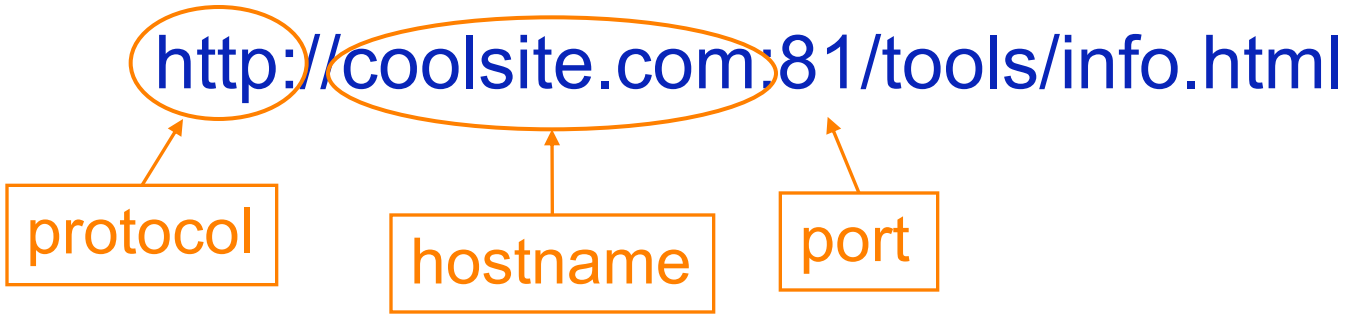
No security barrier

wikipedia.org

wikipedia.org

Origin

- Granularity of protection for same origin policy
- Origin = (protocol, hostname, port)



- It is **string matching**! If these match, it is same origin, else it is not. Even though in some cases, it is logically the same origin, if there is no match, it is not

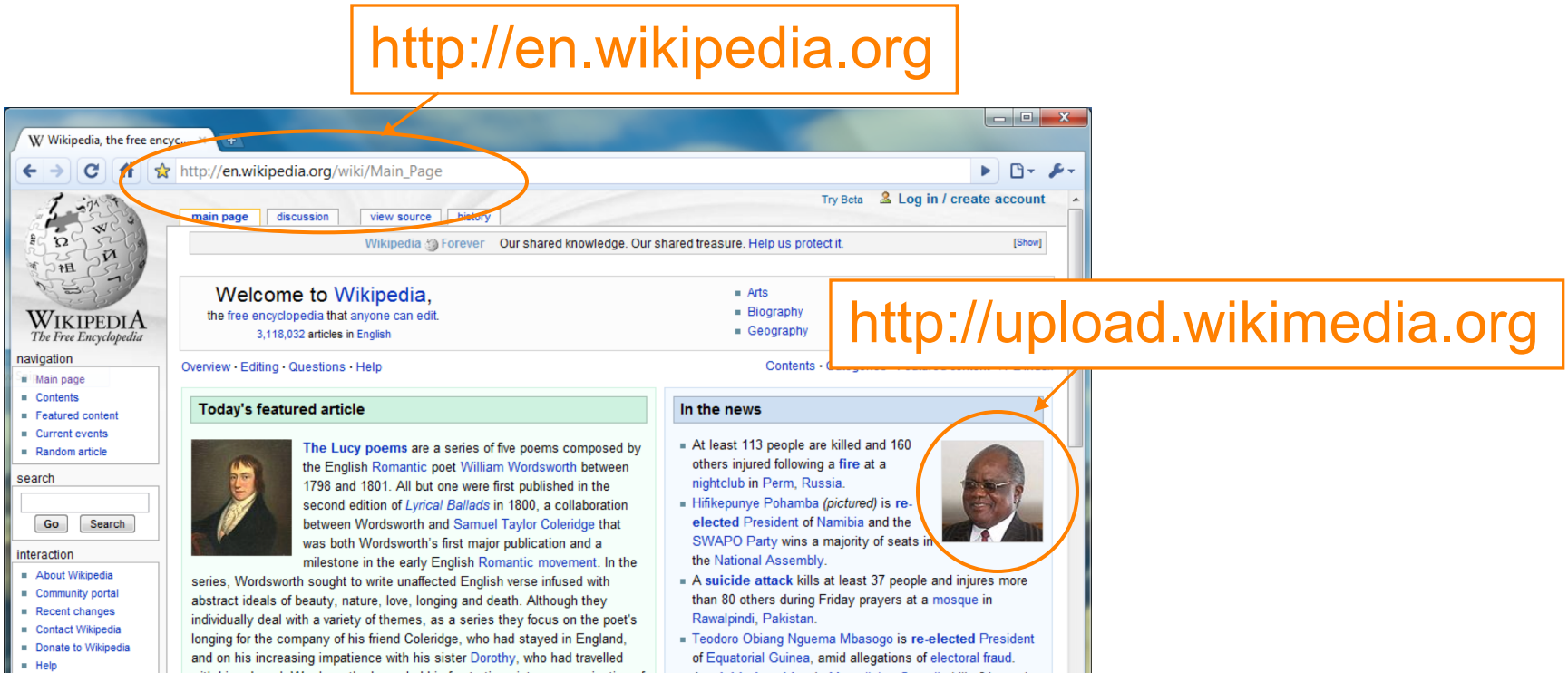
Same-origin policy

One origin should not be able to access the resources of another origin

Javascript on one page cannot read or modify pages from different origins

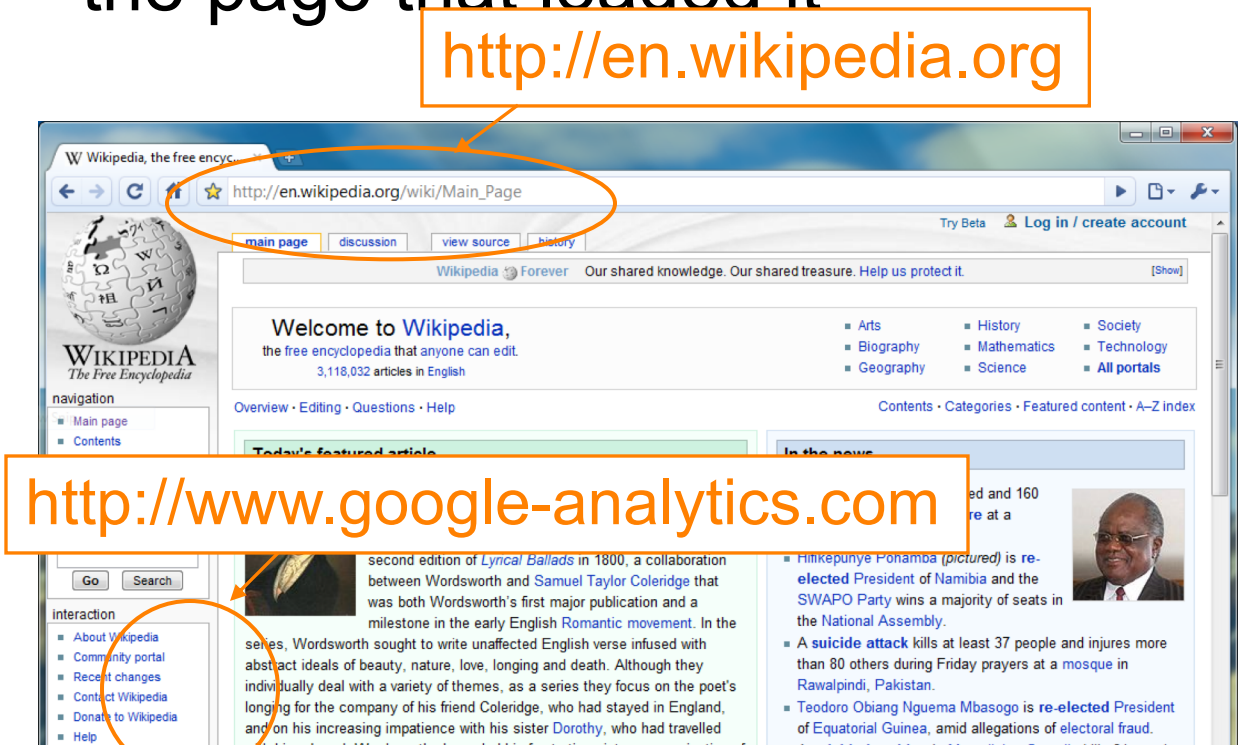
Same-origin policy

- The origin of a page is derived from the URL it was loaded from



Same-origin policy

- The origin of a page is derived from the URL it was loaded from
- Special case: Javascript runs with the origin of the page that loaded it



Origins of other components

- **<img src=“”>** the image is “copied” from the remote server into the new page so it has the origin of the embedding page (like JS) and not of the remote origin
- iframe: origin of the URL from which the iframe is served, and not the loading website.

Exercises

Originating document	Accessed document
http://wikipedia.org/**a**/	http://wikipedia.org/**b**/
http://wikipedia.org/	http://**www.**wikipedia.org/
http://wikipedia.org/	**https**://wikipedia.org/
http://wikipedia.org**:81**/	http://wikipedia.org**:82**/
http://wikipedia.org**:81**/	http://wikipedia.org/

except  !!!

Random fact about … Scott Shenker

Bio according to me: A legend in computer networking

Bio on EECS website: "Scott Shenker spent his academic youth studying theoretical physics but soon gave up chaos theory for computer science. Continuing to display a remarkably short attention span, his research over the years has wandered […]. Unable to focus on any single topic, his current research projects include software-defined networking, […]. Unable to hold a steady job, he currently splits his time between the UC Berkeley Computer Science Division and the ICSI."

Bio from him: "I have never taken a CS course in my life, and I don't program."

If you want to how he succeeded … **attend next lecture**