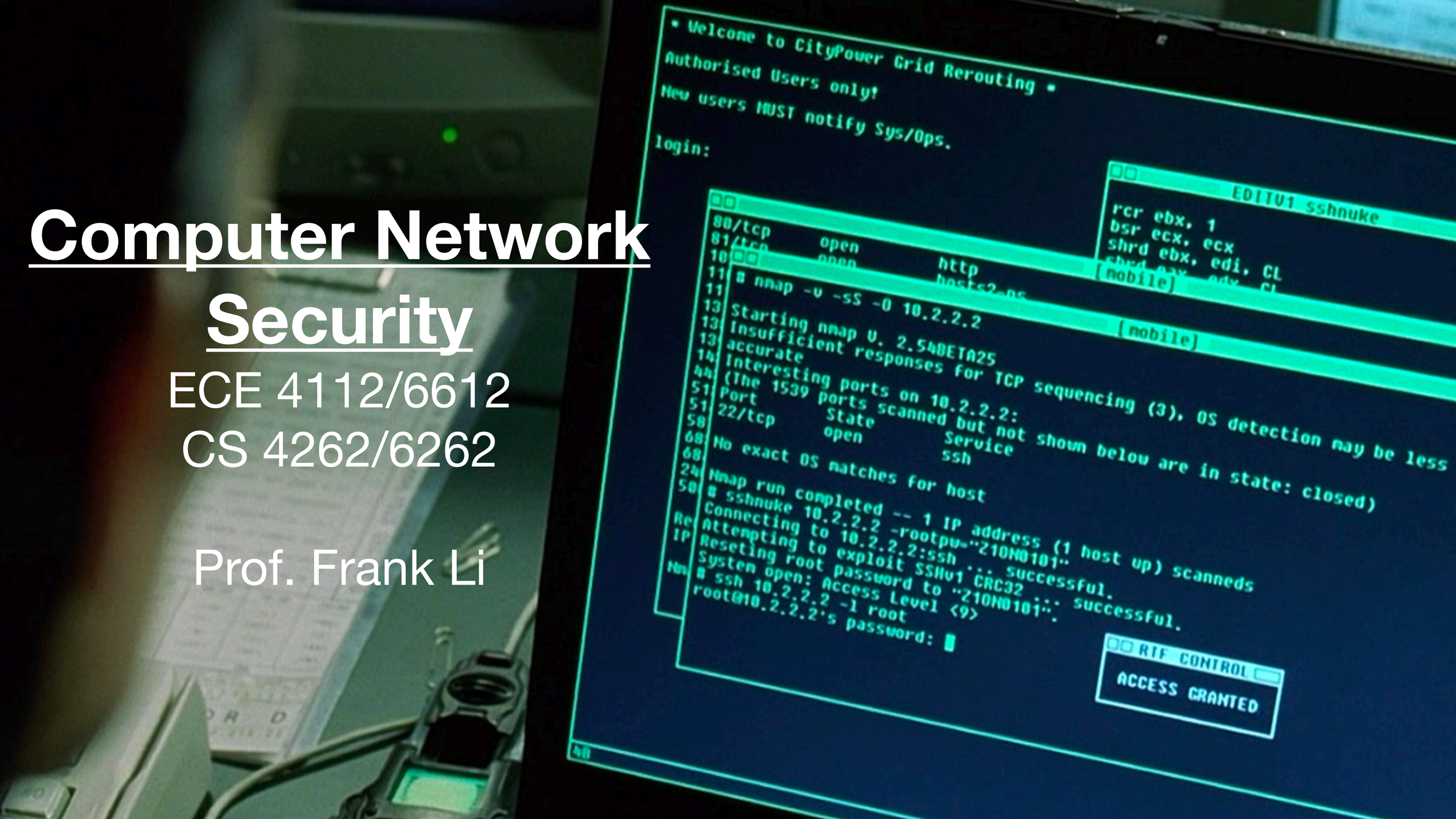
Computer Network Security

ECE 4112/6612
CS 4262/6262

Prof. Frank Li



Logistics

Tue, Oct 10	**No Class (Fall Break)**
Thu, Oct 12	Web security Part 1: Web attacks and defenses
Tue, Oct 17	Web security Part 2: Web attacks and defenses
Thu, Oct 19	Web security Part 3: Web attacks and defenses
Tue, Oct 24*	**Quiz 2**
Thu, Oct 26*	Authentication

HW2 due Tuesday, Oct 17 midnight

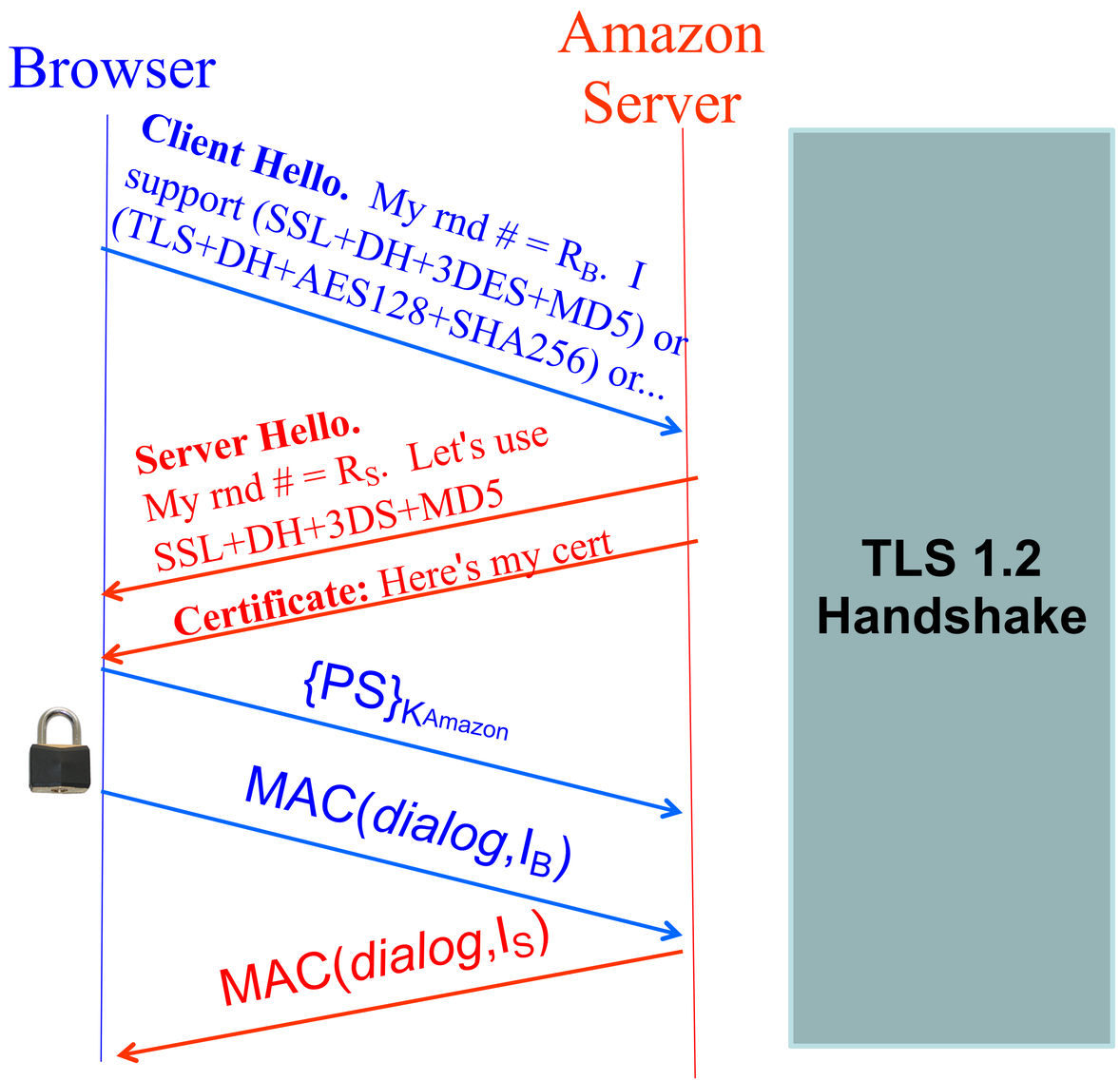
Quiz 2 in 1.5 weeks (Tue, Oct 24) -> No web security, but includes IP + BGP

Project presentation + report assignment to be released (including presentation scheduling)

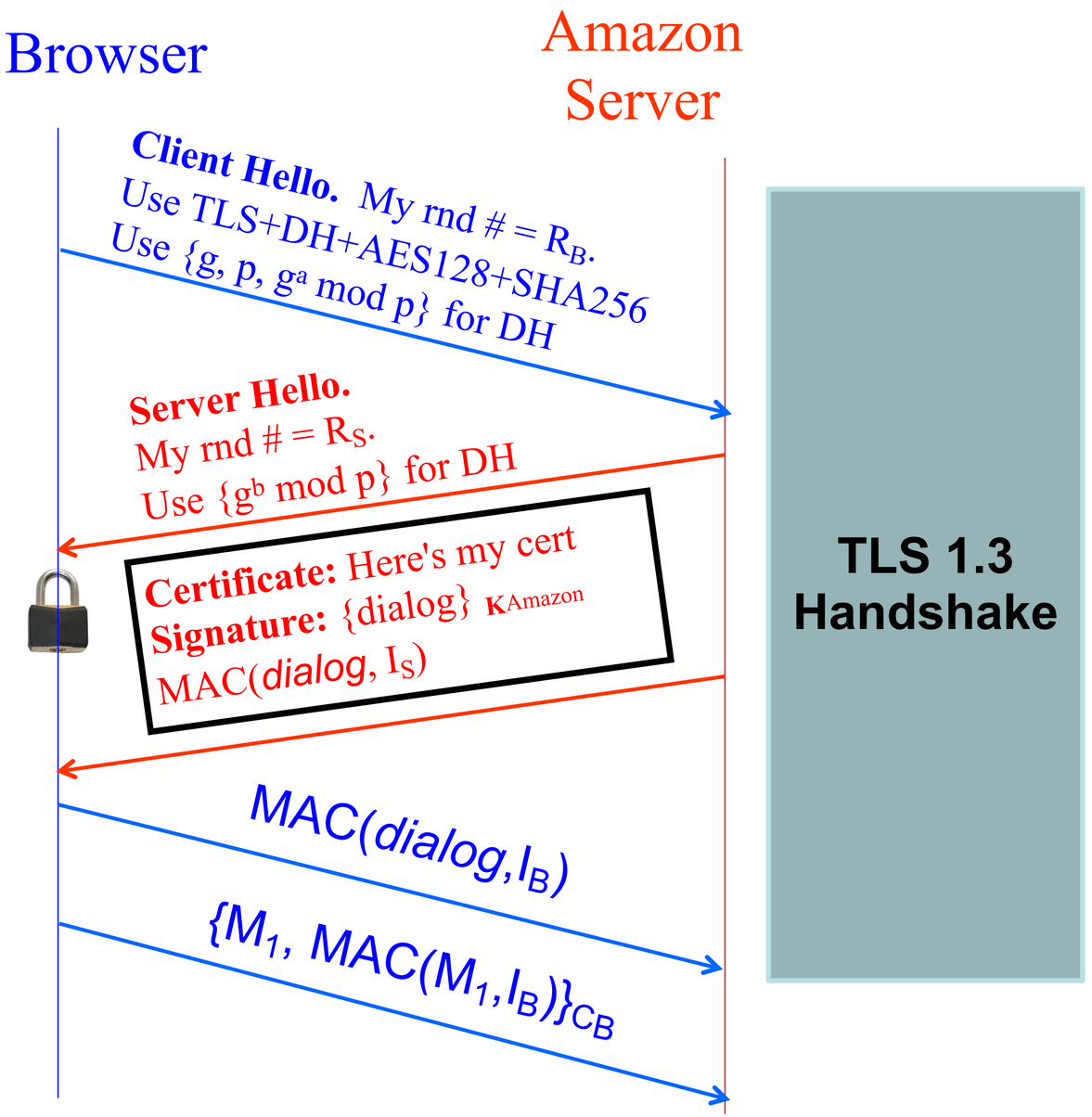
Note: I'll be away the week of Oct 24-26. TAs will proctor Quiz 2, guest lecture for Thu lecture.

Finishing up TLS

TLS 1.2


Browser
Amazon Server
Client Hello. My rnd # = R_B. I support (SSL+DH+3DES+MD5) or (TLS+DH+AES128+SHA256) or...
Server Hello. My rnd # = R_S. Let's use SSL+DH+3DS+MD5
Certificate: Here's my cert
$\{PS\}_{K_{Amazon}}$
MAC(dialog,I_B)
MAC(dialog,I_S)
TLS 1.2 Handshake

TLS 1.3


Browser
Amazon Server
Client Hello. My rnd # = R_B.
Use TLS+DH+AES128+SHA256
Use {g, p, g^a mod p} for DH
Server Hello.
My rnd # = R_S.
Use {g^b mod p} for DH
Certificate: Here's my cert
Signature: {dialog} K_{Amazon}
MAC(dialog, I_S)
MAC(dialog,I_B)
{M_1, MAC(M_1,I_B)}$_{CB}$
TLS 1.3 Handshake

Validating Amazon's Identity

- Browser compares domain *name* in cert w/ URL
- Browser accesses *separate* cert belonging to **issuer**
  - Might be hardwired into the browser - **trusted root CAs!**
  - There could be a *chain* of these …
- Browser applies issuer's public key to verify certificate's signature
  - Validates signature on its own SHA-256 hash of Amazon's cert
- Assuming signature validates, now have high confidence it's indeed Amazon …
  - ***assuming signatory is trustworthy***

End-to-End ⇒ Powerful Protections

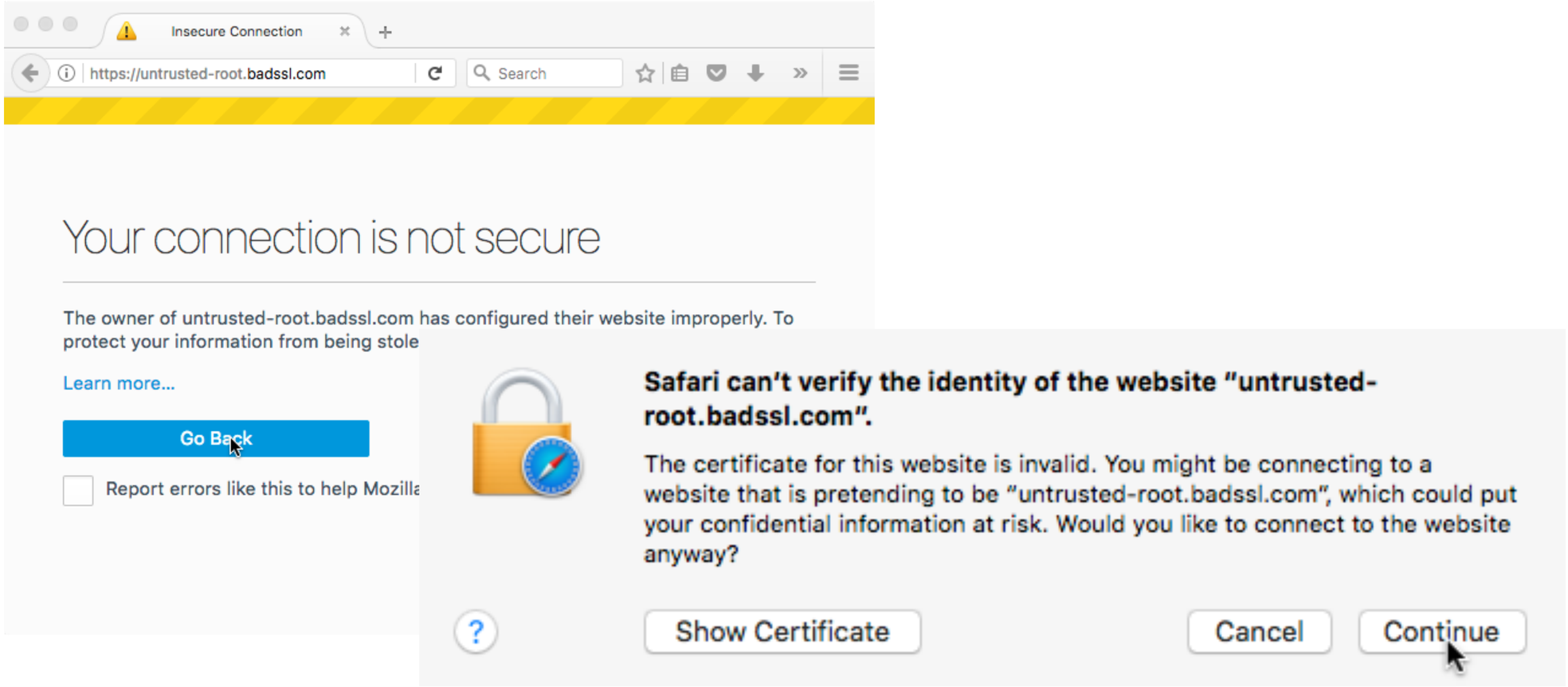
- Attacker runs a sniffer to capture our WiFi session?
  - (maybe by buying a cup of coffee to get the password)
  - But: encrypted application data is unreadable
    - No problem!
- DNS cache poisoning?
  - Client goes to wrong server
  - But: detects impersonation since attacker lacks valid cert
    - No problem!
- Attacker hijacks our connection, injects new traffic
  - But: data receiver rejects it due to failed integrity check
    - No problem!

Powerful Protections, con't

- DHCP spoofing?
  - Client goes to wrong server
  - But: they can't read; we detects impersonation
    - No problem!
- Attacker manipulates BGP routing to run us by an eavesdropper or take us to the wrong server?
  - But: they can’t read; we detect impersonation
    - No problem!
- Attacker slips in as a Man In The Middle?
  - But: they can’t read, they can’t inject
  - They can’t even replay previous encrypted traffic
  - **No problem!**

Validating Amazon's Identity, con't

- What if browser can't find a cert for the issuer or the cert doesn't pass?



Validating Amazon's Identity, con't

- What if browser can't find a cert for the issuer or the cert doesn't pass?
  - Then warns the user that site has not been verified
    - Note, can still proceed, just without authentication
- Q: Which end-to-end security properties do we lose if we incorrectly trust that the site is whom we think?
- A: **All of them!**
  - Goodbye confidentiality, integrity, authentication
  - Attacker can read everything, modify, impersonate

How do we know to use TLS?

- How do we know if example.com is over HTTP or HTTPS?
  - Method 1: Could try HTTPS, and if we don't get any response or an error, switch to HTTP.
  - Method 2: Could try HTTP, and if HTTPS is available, the server's HTTP response can redirect us to HTTPS

How do we know to use TLS?

- What if there's a network MITM attacker??
  - W/ Method 1: MITM can block/drop HTTPS attempt, causing us to switch to HTTP
  - W/ Method 2: Block the HTTP redirection response so we don't know HTTPS is available

  - In both cases, then MITM attacker can pretend to be the HTTP server. We will send our data to the attacker over HTTP and the attacker can relay data to the real HTTPS website.
  - **SSL-Stripping attack**

How do we know to use TLS?

- Solutions?
  - **HTTP Strict Transport Security (HSTS):** Server tells client that it always uses HTTPS, so client knows to use HTTPS moving forward.
    - Done using an HTTP header

```
Strict-Transport-Security: max-age=31536000;
```

    - Assumes secure first connection though...

  - **(Proposed) HTTPS DNS Resource Records:** Add a new DNS RR indicating HTTPS is available, which clients can query for.
    - Unless DNSSEC used, may not help in MITM case

TLS Limitations

- Properly used, TLS provides powerful end-to-end protections
- So why not use it for *everything*??
- Issues:
  - Cost of public-key crypto
    - Takes non-trivial CPU processing (but today a minor issue)
    - Note: *symmetric* key crypto on modern hardware is non-issue
  - Hassle of buying/maintaining certs (fairly minor)


https://letsencrypt.org
let's encrypt
LINUX FOUNDATION COLLABORATIVE PROJECTS
Let's Encrypt
Documentation
Get Help
Donate
About Us
Let's Encrypt is a free, automated, and open Certificate Authority.
Get Started
Donate
FROM OUR BLOG
Mar 23, 2017
OVH Renews Platinum Sponsorship of Let's Encrypt
We're pleased to announce that OVH has renewed their support for Let's Encrypt as a Platinum sponsor for the next three years.
Read more
MAJOR SPONSORS
mozilla
Akamai
CISCO
EFF
OVH
chrome
IdenTrust
gemalto
FORD FOUNDATION
facebook
AUTOMATTIC
shopify

TLS Limitations

- Properly used, TLS provides powerful end-to-end protections
- So why not use it for *everything*??
- Issues:
  - Cost of public-key crypto
    - Takes non-trivial CPU processing (but today a minor issue)
    - Note: *symmetric* key crypto on modern hardware is non-issue
  - Hassle of buying/maintaining certs (fairly minor)
  - **Latency**: extra round trips ⇒ pages take longer to load

TLS Limitations, con't

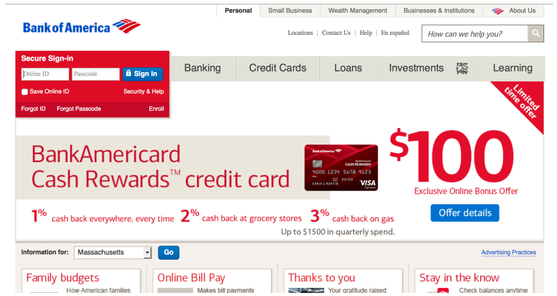
- Problems that TLS does **not** take care of ?
- TCP-level attacks
  - RST injection
    - (but does protect against data injection!)
- Denial of Service Attacks
- Application-level vulnerabilities:
  - SQL injection / XSS / server-side coding/logic flaws
  - Browser coding/logic flaws
  - User flaws
    - Weak passwords
    - Phishing
  - HTTP server vulnerabilities

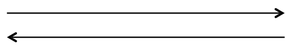
Web Security
(Part 1)

What is the Web?

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

client browser





web server

Bank of America

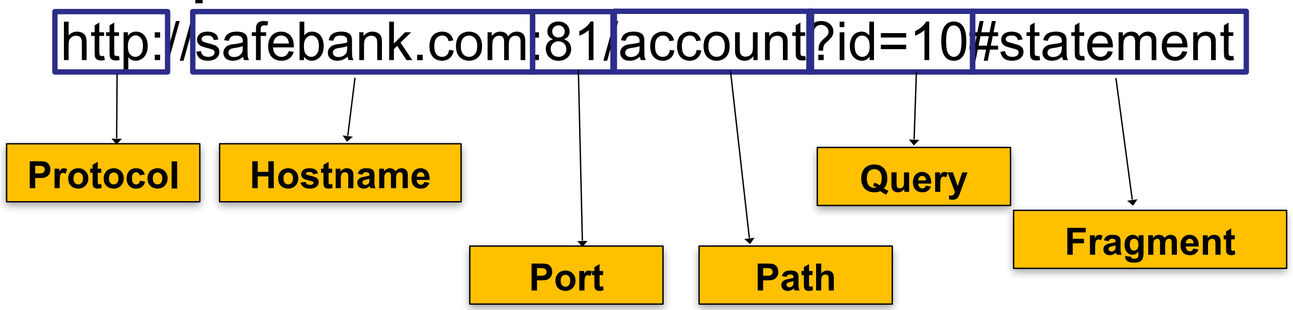
HTTP
(Hypertext Transfer Protocol)

A common data communication protocol on the web

URLs

Global identifiers of network-retrievable resources

Example:



HTTP

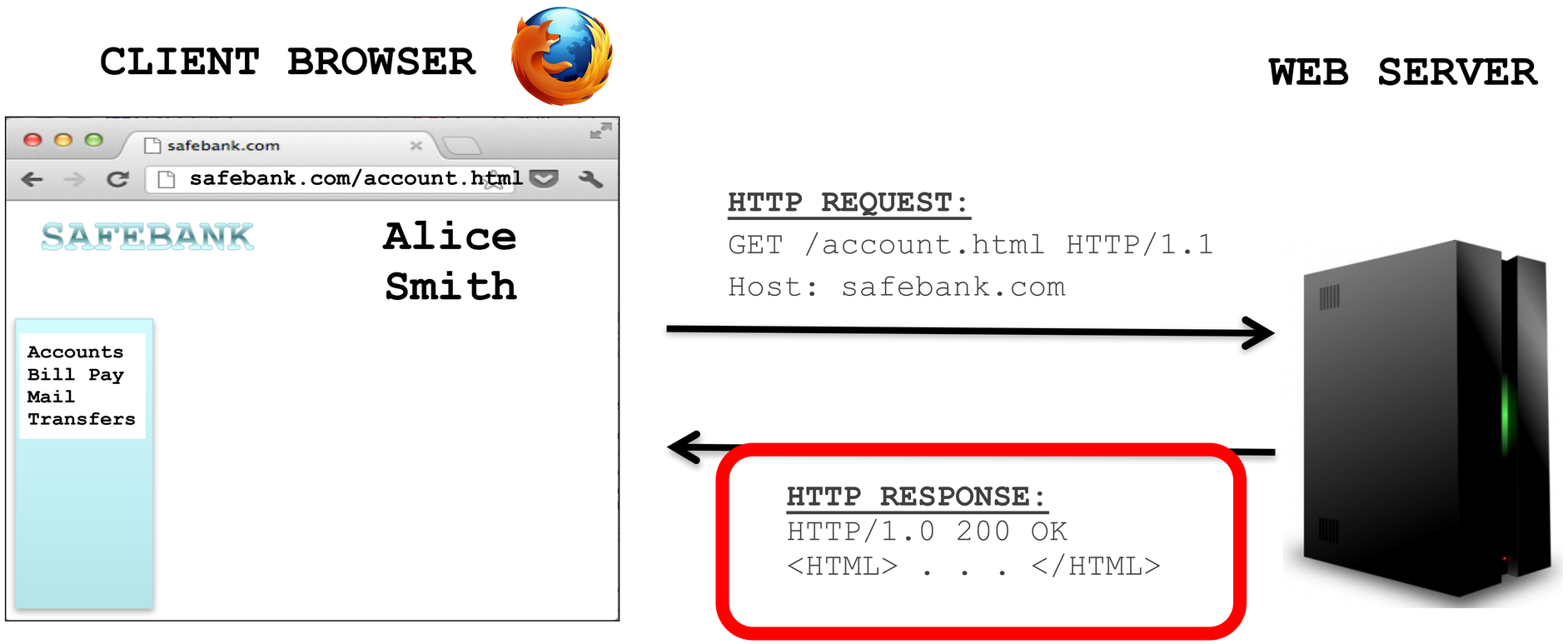

CLIENT BROWSER
WEB SERVER
safebank.com
safebank.com/account.html
SAFEBANK
Alice
Smith
Accounts
Bill Pay
Mail
Transfers
HTTP REQUEST:
GET /account.html HTTP/1.1
Host: safebank.com
HTTP RESPONSE:
HTTP/1.0 200 OK
<HTML> . . . </HTML>

HTTP Request

GET: no side effect (supposedly)

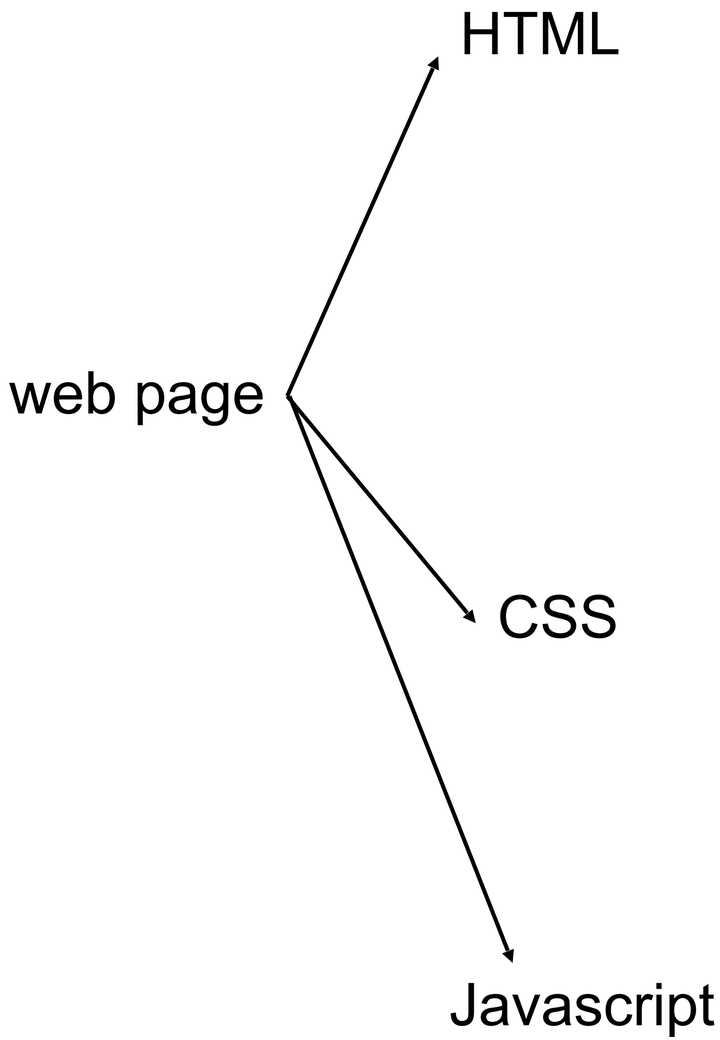
POST: possible side effect, includes additional data

HTTP


CLIENT BROWSER
WEB SERVER
safebank.com
safebank.com/account.html
SAFEBANK
Alice
Smith
Accounts
Bill Pay
Mail
Transfers
HTTP REQUEST:
GET /account.html HTTP/1.1
Host: 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)

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>
```

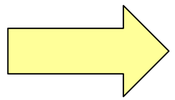
Very powerful!

DOM (Document Object Model)

Cross-platform model for representing and interacting with objects in HTML

HTML

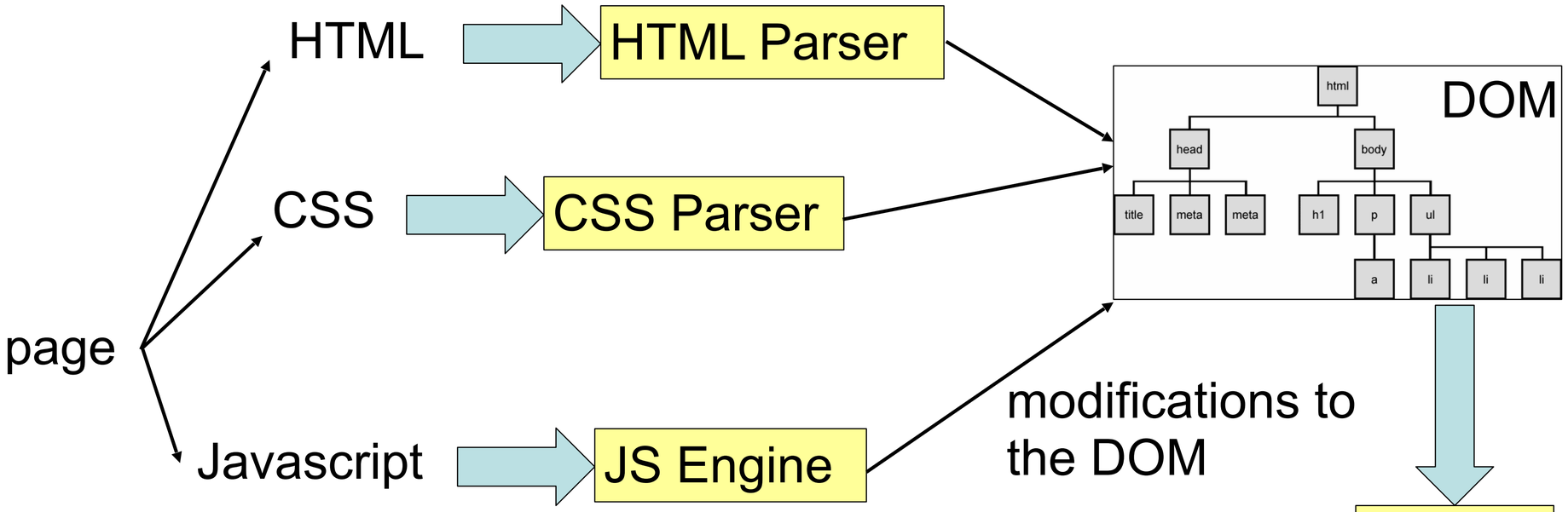
```
<html>
    <body>
        <div>
            foo
        </div>
        <form>
            <input type="text" />
            <input type="radio" />
            <input type="checkbox" />
>
        </form>
    </body>
</html>
```

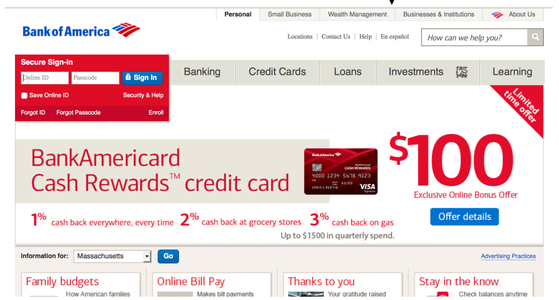


DOM Tree

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

Page rendering





What can you do with Javascript?

Almost anything you want to the DOM!

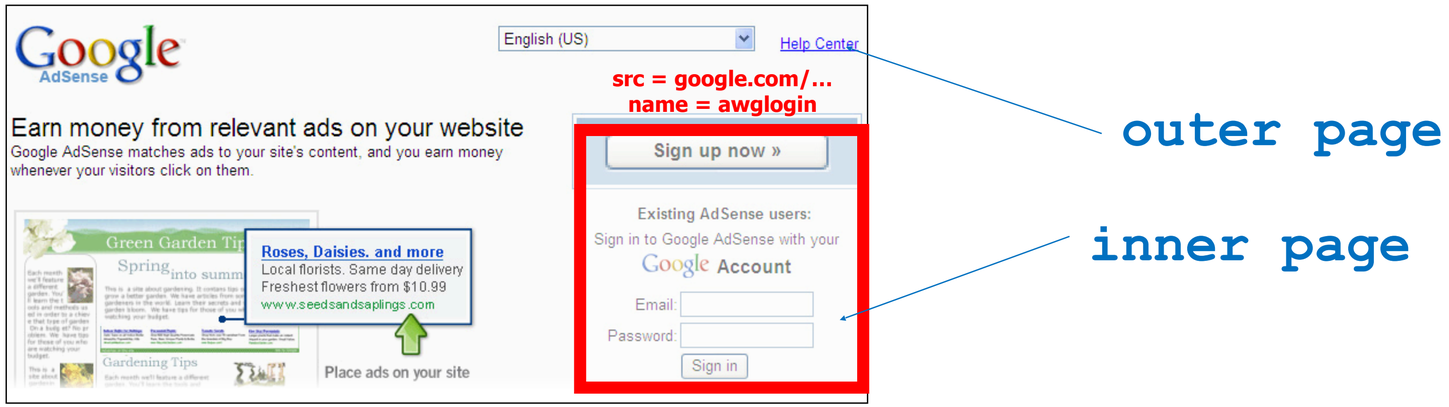
A JS script embedded on a page can modify
in almost arbitrary ways the DOM of the page.

The same happens if an attacker manages to
get you load a script into your page.

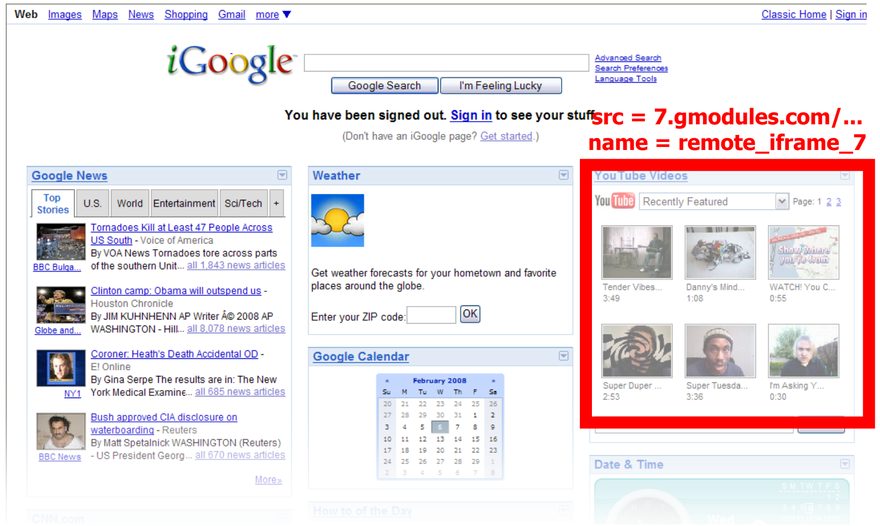
Frames

- 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 inside its own rectangle

Frames

- Outer page can specify only sizing and placement of the frame in the outer page
- Frame isolation: Outer page cannot access contents of inner page; inner page cannot access contents of outer page (unless both pages are from the same site*)

Outrageous Chocolate Chip Cookies

4.5 stars 1676 reviews

Made 321 times

Recipe by: Joan

"A great combination of chocolate chips, oatmeal, and peanut butter."

HTTP cookies

Print

Ingredients

25 m | 18 servings | 207 cals

- 1/2 cup butter
- 1/2 cup white sugar
- 1/3 cup packed brown sugar
- 1 cup all-purpose flour
- 1 teaspoon baking soda
- 1/4 teaspoon salt
- 1/2 cup rolled oats
- 1 cup semisweet chocolate chips

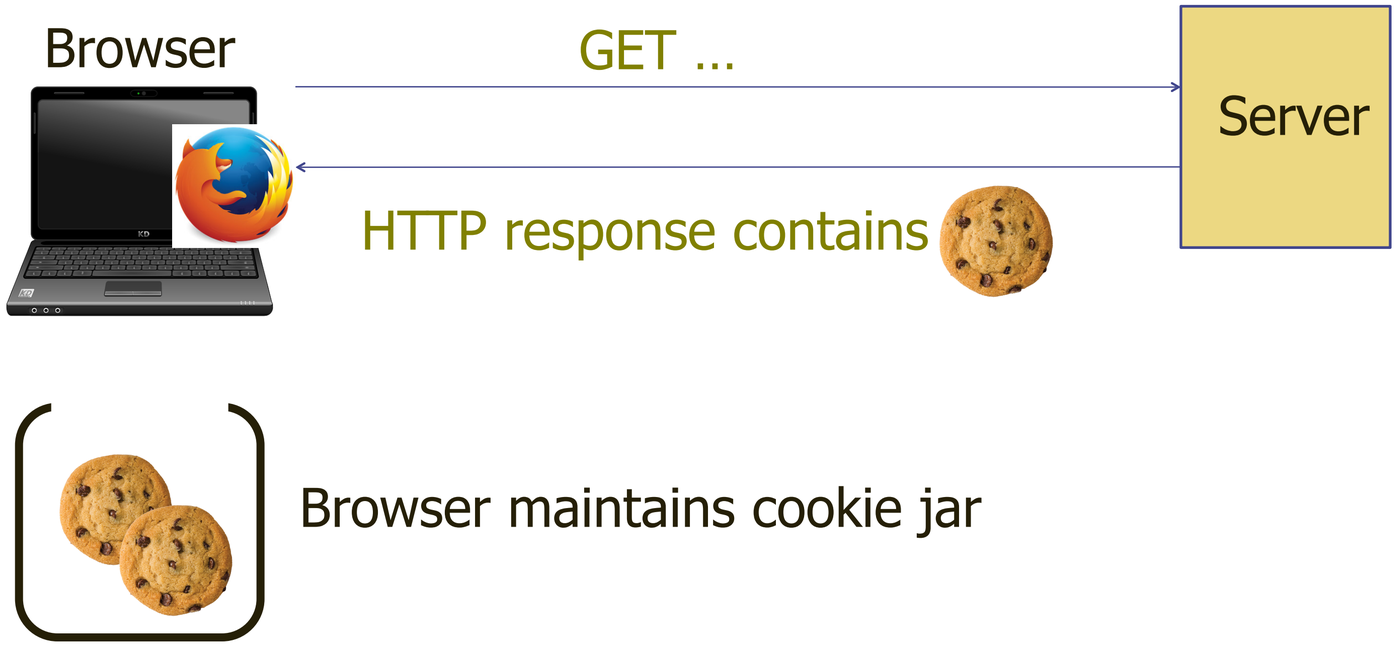
On Sale

On

What's on sale near you.

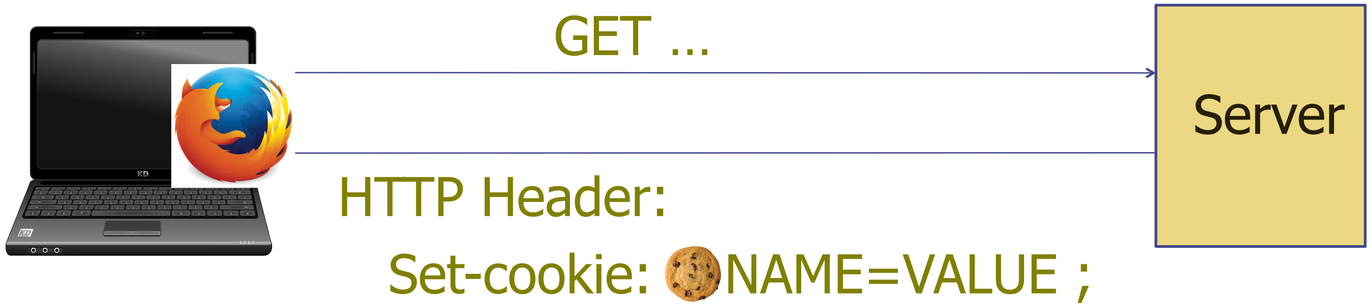
Cookies

- A way of maintaining state



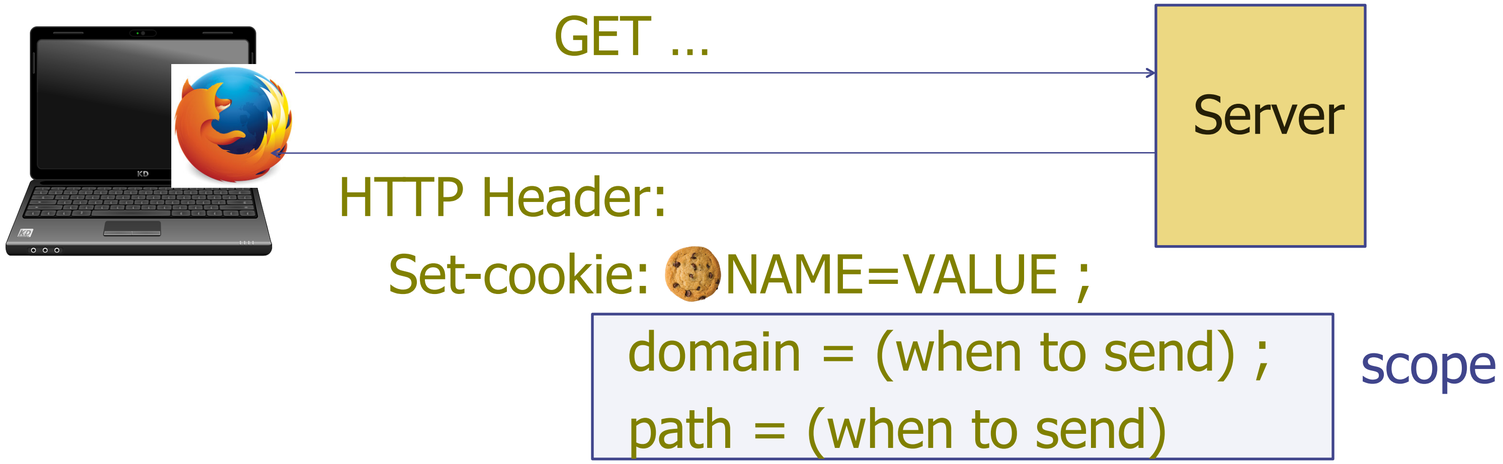
Browser maintains cookie jar

Setting/deleting cookies by server



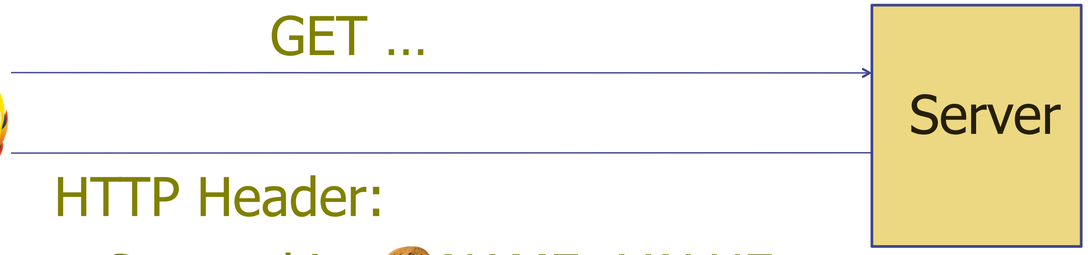
- The first time a browser connects to a particular web server, it has no cookies for that web server
- When the web server responds, it includes a **Set-Cookie:** header that defines a cookie
- Each cookie is just a name-value pair

Cookie scope



- When the browser connects to the same server later, it includes a Cookie: header containing the name and value, which the server can use to connect related requests.
- Domain and path inform the browser about which sites to send this cookie to

Cookie scope



HTTP Header:

Set-cookie: NAME=VALUE ;

domain = (when to send) ;

path = (when to send)

secure = (only send over HTTPS);

- Secure: sent over HTTPS only
  - HTTPS provides secure communication (privacy, authentication, integrity)

Cookie scope

GET …

Server

HTTP Header:

Set-cookie: NAME=VALUE ;

domain = (when to send) ;

path = (when to send)

secure = (only send over HTTPS);

expires = (when expires) ;

HttpOnly

- Expires is expiration date
- HttpOnly: cookie cannot be accessed by Javascript, but only sent by browser

Client side read/write: document.cookie

- Setting a cookie in Javascript:

  document.cookie = "name=value; expires=…; "

- Reading a cookie: alert(document.cookie)

  prints string containing all cookies available for document (based on [protocol], domain, path)

- Deleting a cookie:

  document.cookie = "name=; expires= Thu, 01-Jan-70"

document.cookie often used to customize page in Javascript

Cookie scope

- Scope of cookie might not be the same as the URL-host name of the web server setting it

Rules on:

1. What scopes a URL-hostname (server) is allowed to set
2. When a cookie is sent to a URL

What scope a server may set for a cookie

The browser checks if the server may set the cookie, and if not, it will not accept the cookie.

domain: any domain-suffix of URL-hostname, except TLD

[top-level domains, e.g. '.com']

example: host = "login.site.com"

allowed domains	disallowed domains
login.site.com	**user.site.com**
.site.com	**othersite.com**
	.com

⇒ Cookies are sent to domain + subdomains

⇒ **login.site.com** can set cookies for all of **.site.com** but not for another site or TLD

Problematic for sites like .gatech.edu

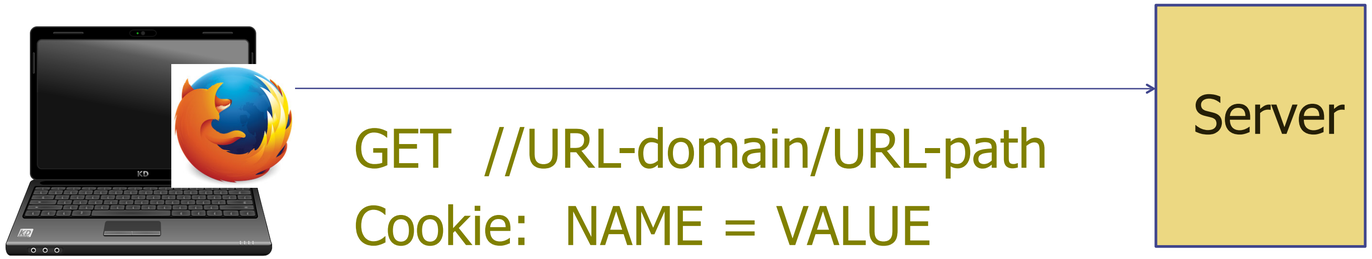
path: can be set to anything

Examples

Web server at foo.example.com wants to set cookie with domain:

Domain	Whether it will be set, and if so, where it will be sent to
[value omitted]	foo.example.com
bar.foo.example.com	Cookie not set; more specific than original domain
foo.example.com	*. foo.example.com
baz.example.com	Cookie not set; domain mismatch
example.com	*.example.com
ample.com	Cookie not set; domain mismatch
.com	Cookie not set; TLD is too broad

When browser sends cookie

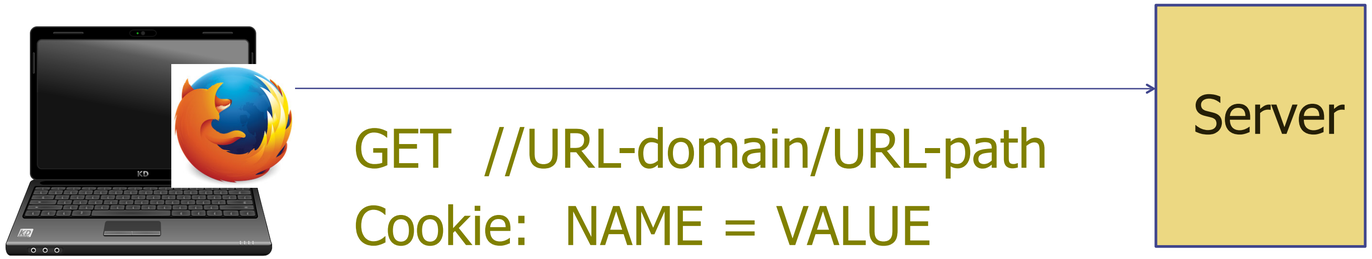


Goal: server only sees cookies in its scope

Browser sends all cookies in URL scope:

- cookie-domain is domain-suffix of URL-domain, and
- cookie-path is prefix of URL-path, and
- [protocol=HTTPS if cookie is “secure”]

When browser sends cookie



A cookie with

domain = example.com, and

path = /some/path/

will be included on a request to

http://foo.example.com/some/path/subdirectory/hello.txt

Examples: Which cookie will be sent?

cookie 1
name = **userid**
value = u1
domain = **login.site.com**
path = **/**
non-secure

cookie 2
name = **userid**
value = u2
domain = **.site.com**
path = **/**
non-secure

http://checkout.site.com/ cookie: userid=u2

http://login.site.com/ cookie: userid=u1, userid=u2

http://othersite.com/ cookie: none

Examples

cookie 1
name = **userid**
value = u1
domain = **login.site.com**
path = **/**
secure

cookie 2
name = **userid**
value = u2
domain = **.site.com**
path = **/**
non-secure

http://checkout.site.com/ cookie: userid=u2

http://login.site.com/ cookie: userid=u2

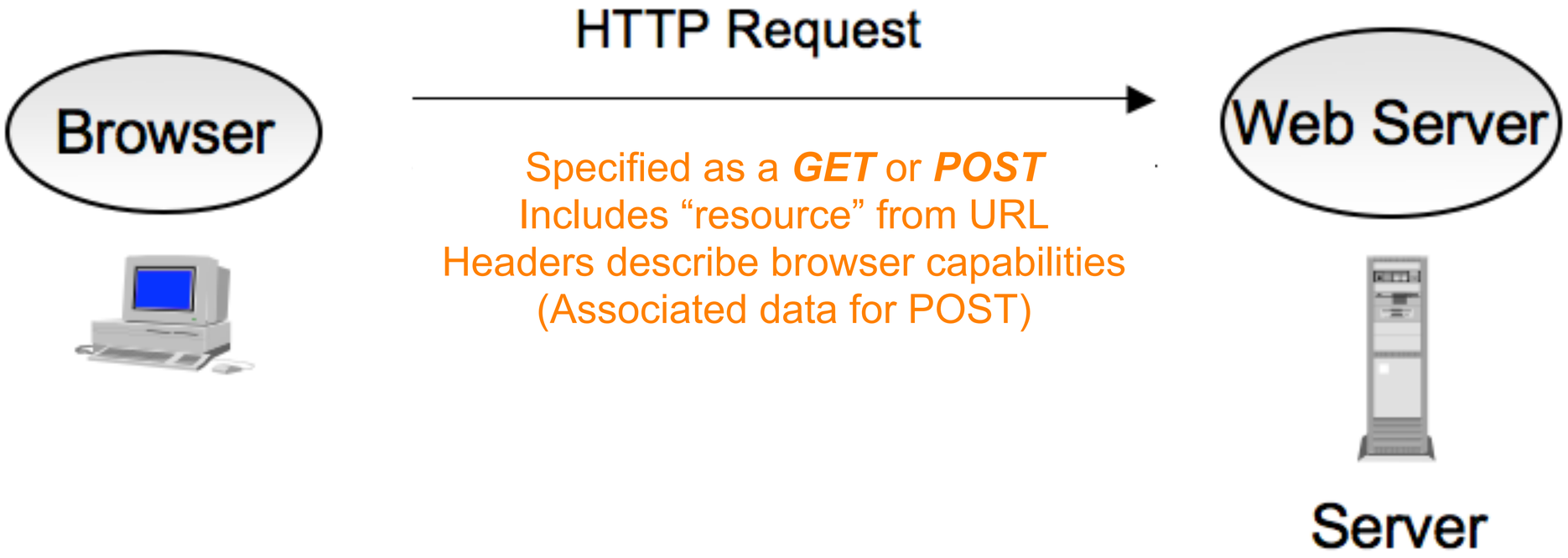
https://login.site.com/ **cookie: userid=u1; userid=u2**

(arbitrary order)

Cookies & Web Authentication

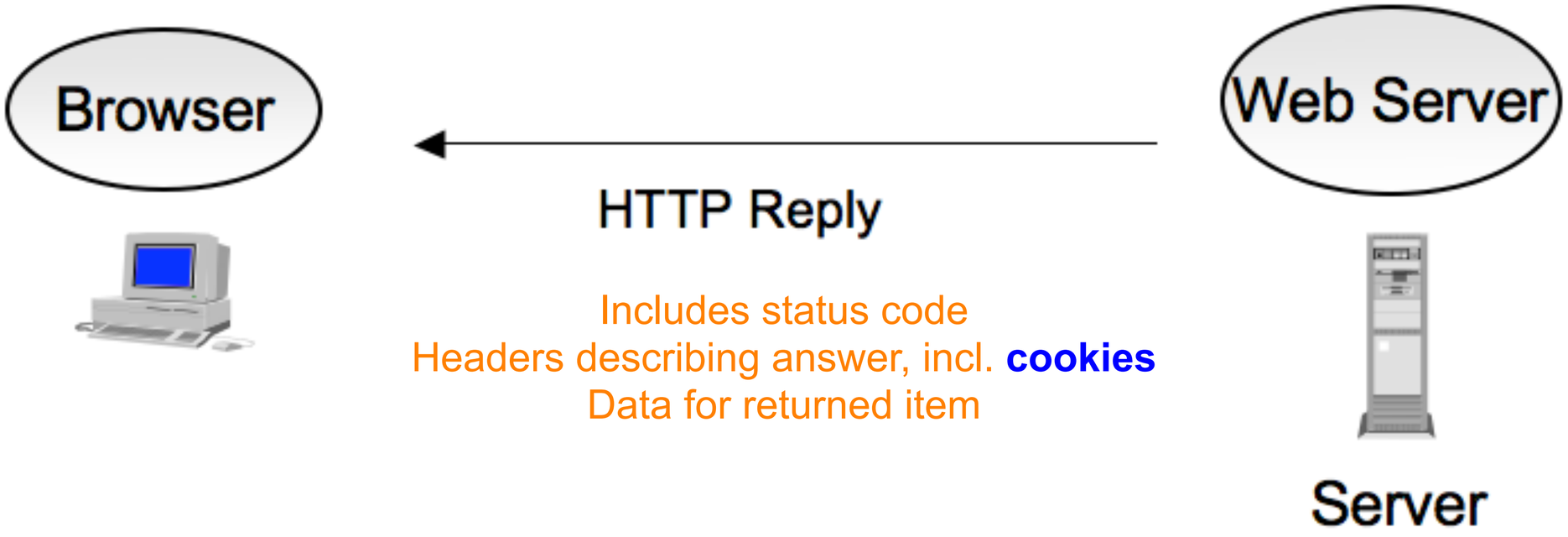
- One very widespread use of cookies is for web sites to track users who have authenticated
- E.g., once browser fetched ***http://mybank.com/login.html?user=alice&pass=bigsecret***
  with a correct password, server associates value of "**session**" cookie with logged-in user's info
  - An "authenticator"

Basic Structure of Web Traffic

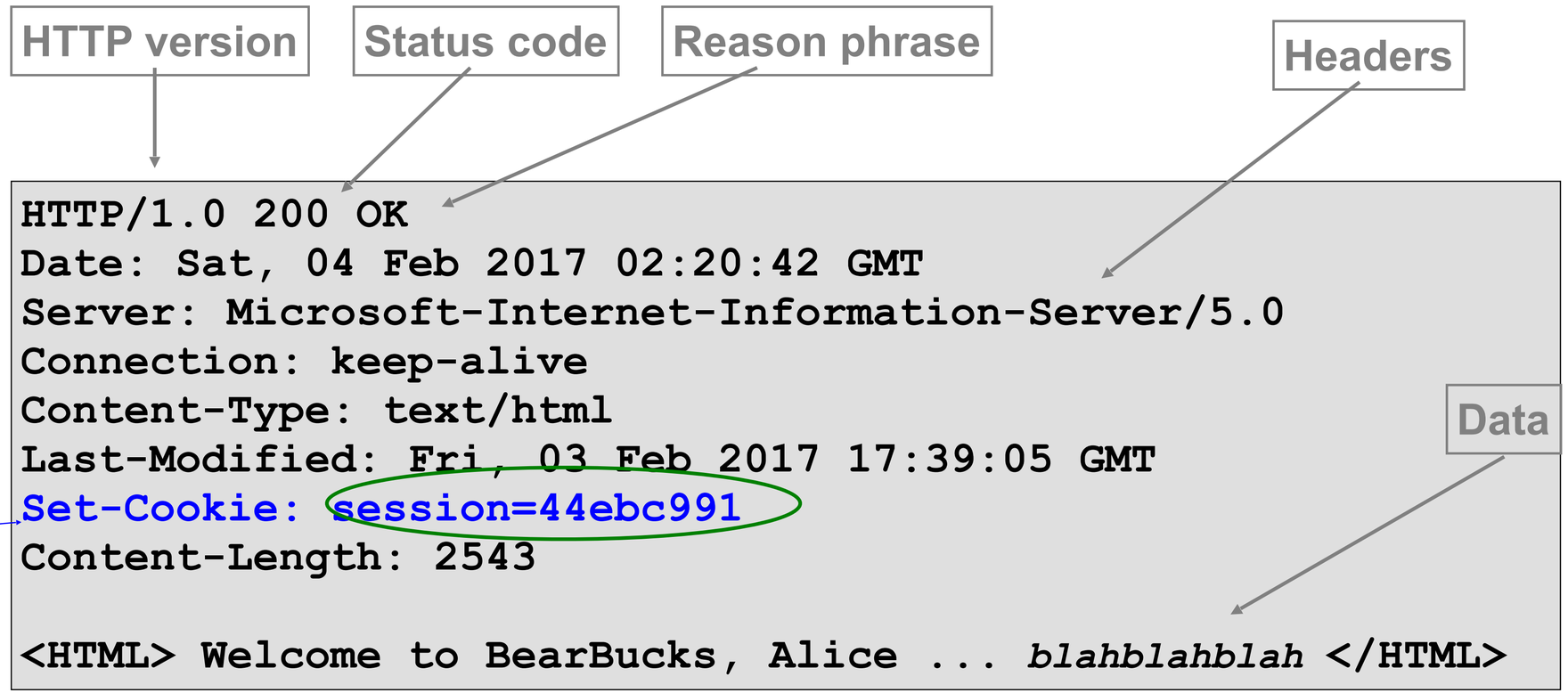


E.g., user clicks on URL:
http://mybank.com/login.html?user=alice&pass=bigsecret

HTTP Cookies

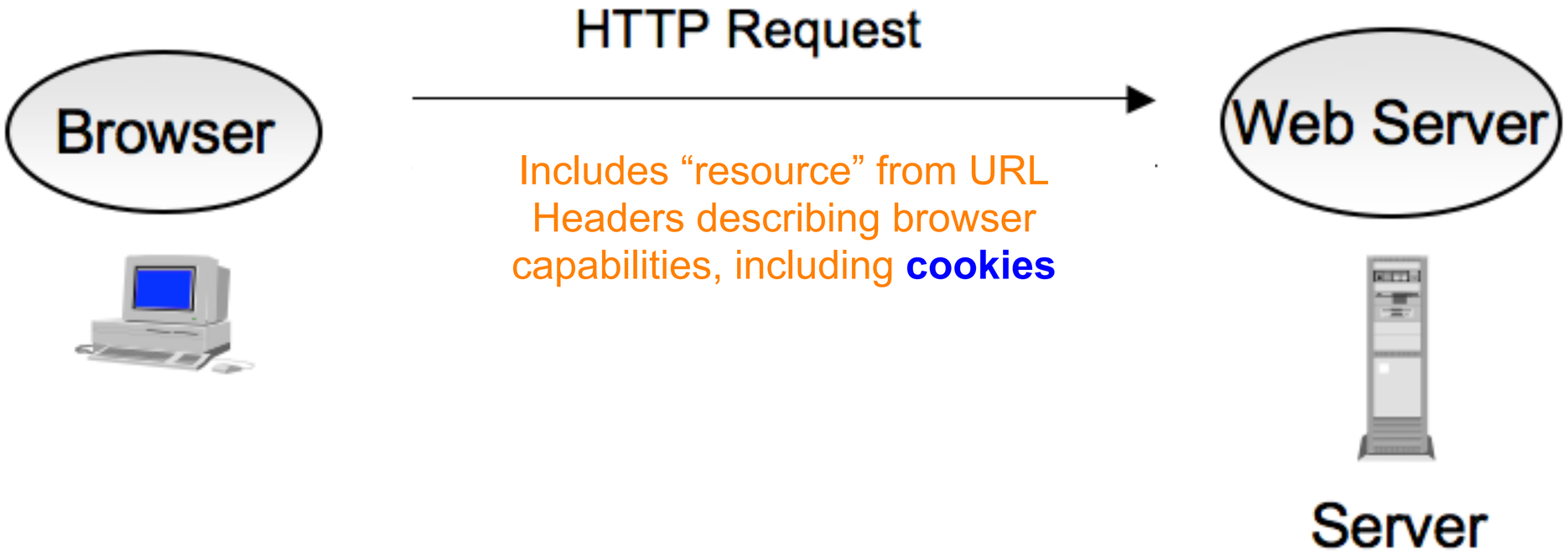


HTTP Response



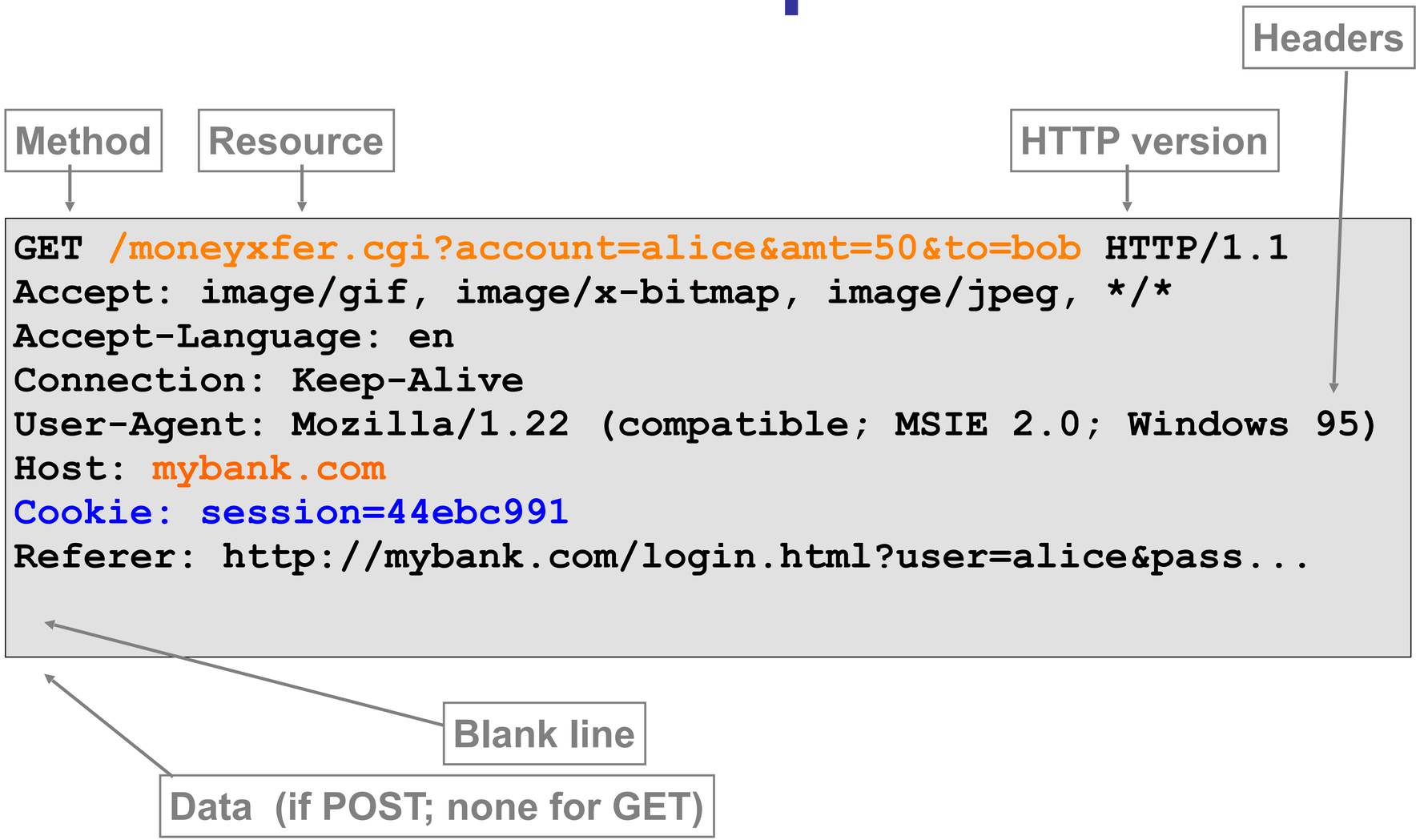
Here the server instructs the browser to remember the cookie "session" so it & its value will be included in subsequent requests

Cookies & Follow-On Requests



E.g., Alice clicks on URL:
http://mybank.com/moneyxfer.cgi?account=alice&amt=50&to=bob

HTTP Request



Cookies & Web Authentication

- One very widespread use of cookies is for web sites to track users who have authenticated
- E.g., once browser fetched ***`http://mybank.com/login.html?user=alice&pass=bigsecret`***
  with a correct password, server associates value of "**`session`**" cookie with logged-in user's info
  - An "authenticator"
- Now server subsequently can tell: "I'm talking to same browser that authenticated as Alice earlier"

`"Cookie theft"`

⇒ *An attacker who can get a copy of Alice's cookie can access the server impersonating Alice!*

Thinking about Web Security

Desirable security goals

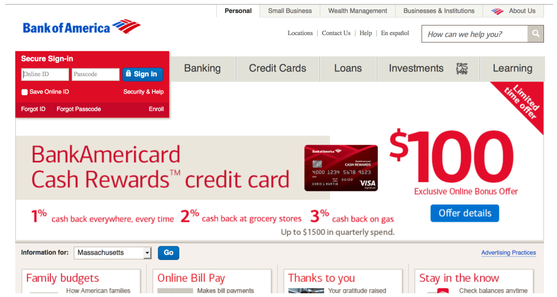
- **Integrity**: malicious web sites should not be able to tamper with integrity of our computers or our information on other web sites
- **Confidentiality/Privacy**: malicious web sites should not be able to learn confidential information from our computers or other web sites
- **Availability**: malicious websites should not be able to keep us from accessing other web resources

Same-origin policy

- Each site in the browser is isolated from all others

browser:

security barrier



wikipedia.org

bankofamerica.com

Same-origin policy

- Multiple pages from the same site are not isolated

browser:

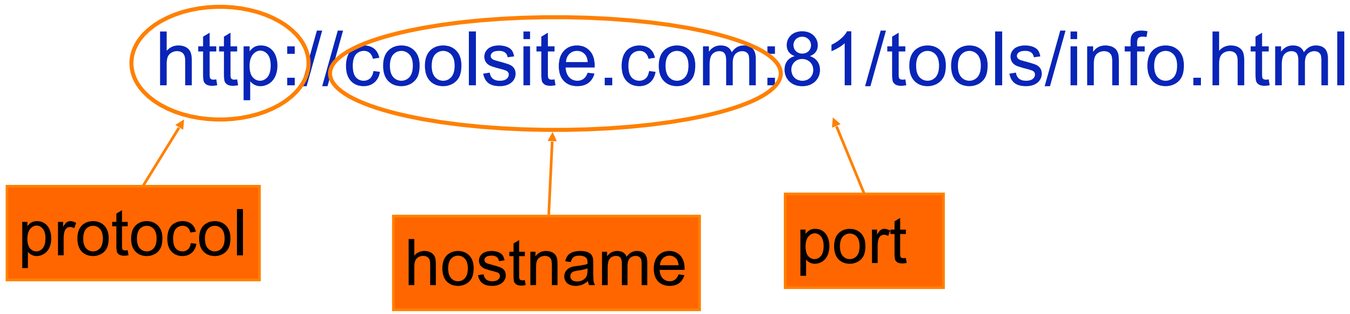
No security barrier

wikipedia.org

wikipedia.org

Origin

- Granularity of protection for same origin policy
- Origin = protocol + hostname + port



- Hostname compared using ***string matching***! If these match (along with protocol + port), it is same origin; else it is not. Even though in some cases, it is logically the same origin (e.g., google.com vs google.uk), if there is no string match, it is not.

Same-origin policy

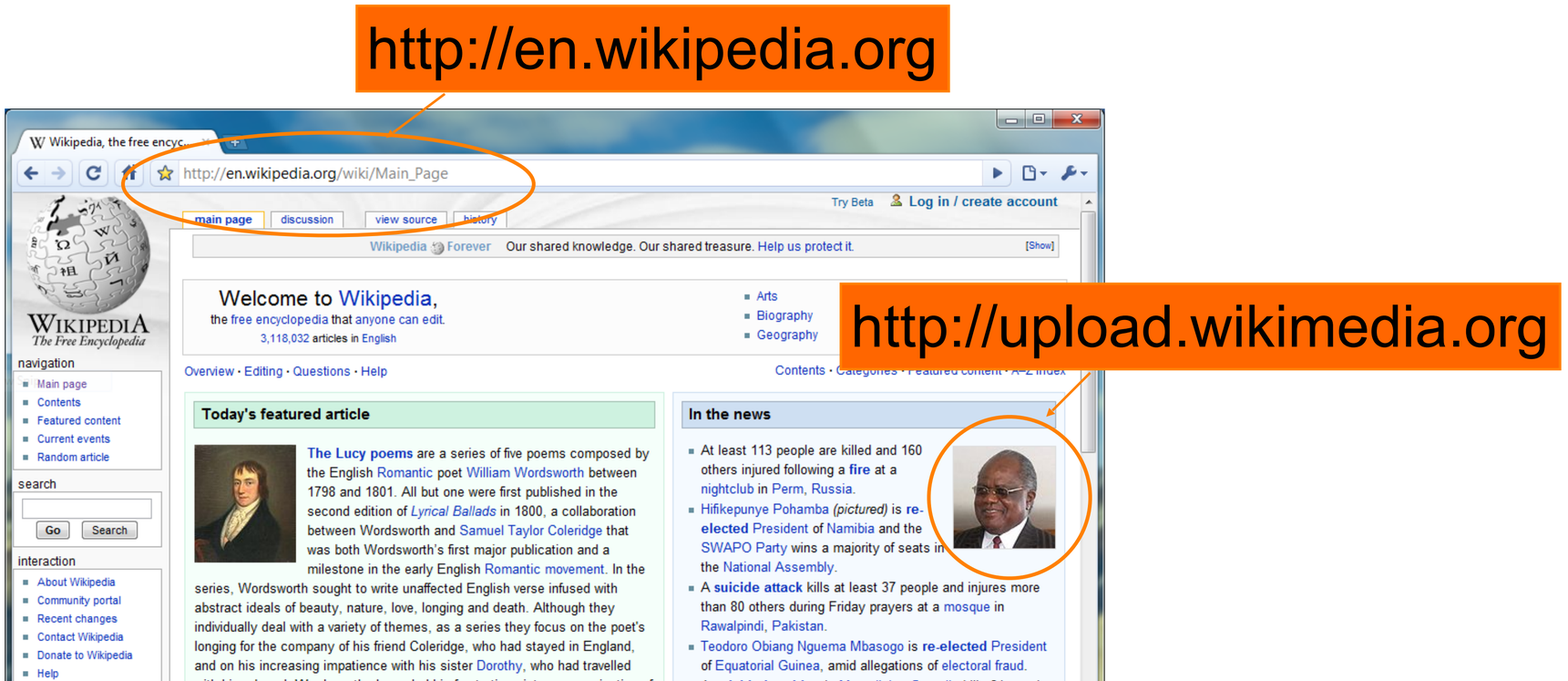
One origin should not be able to access the resources of another origin

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

The contents of an *iframe* have the origin of the URL from which the iframe is served; *not* the loading website.

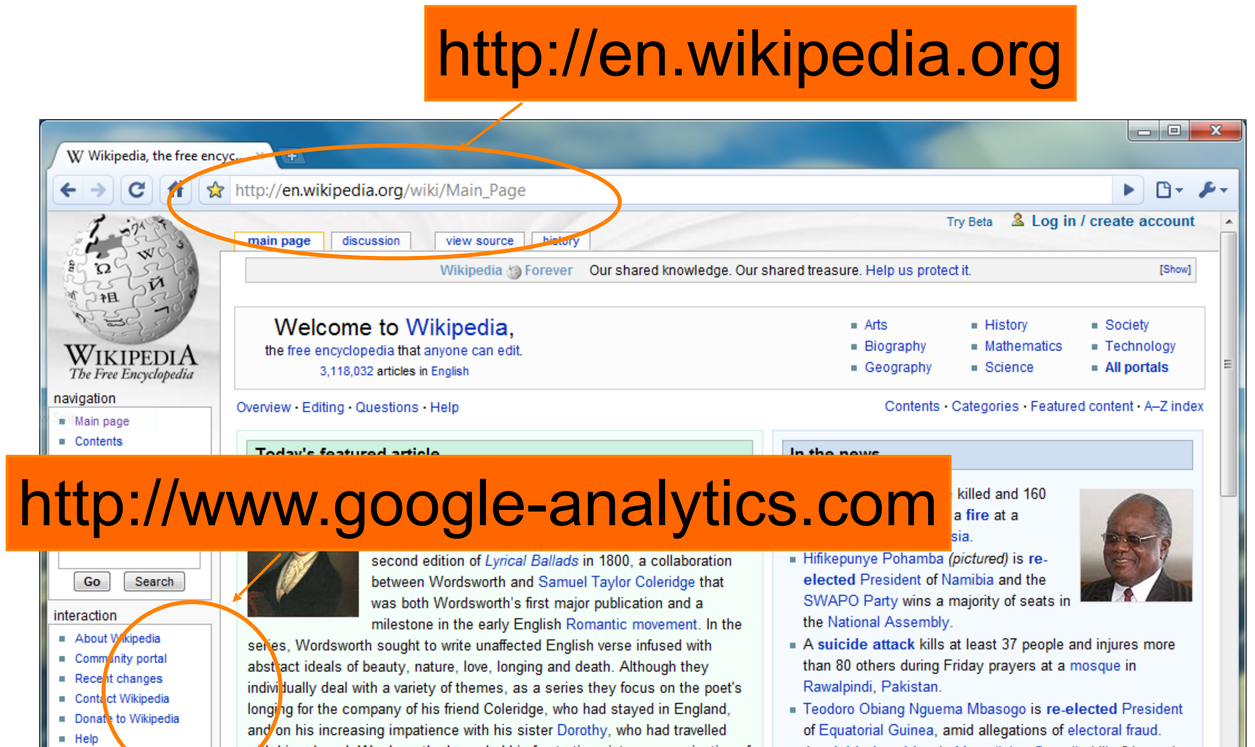
Same-origin policy

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



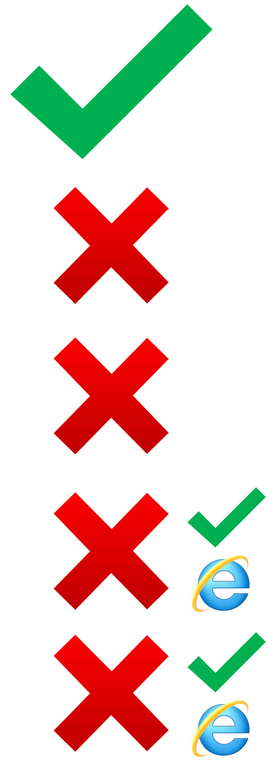
Same-origin policy

- 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



Assessing SOP

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 !