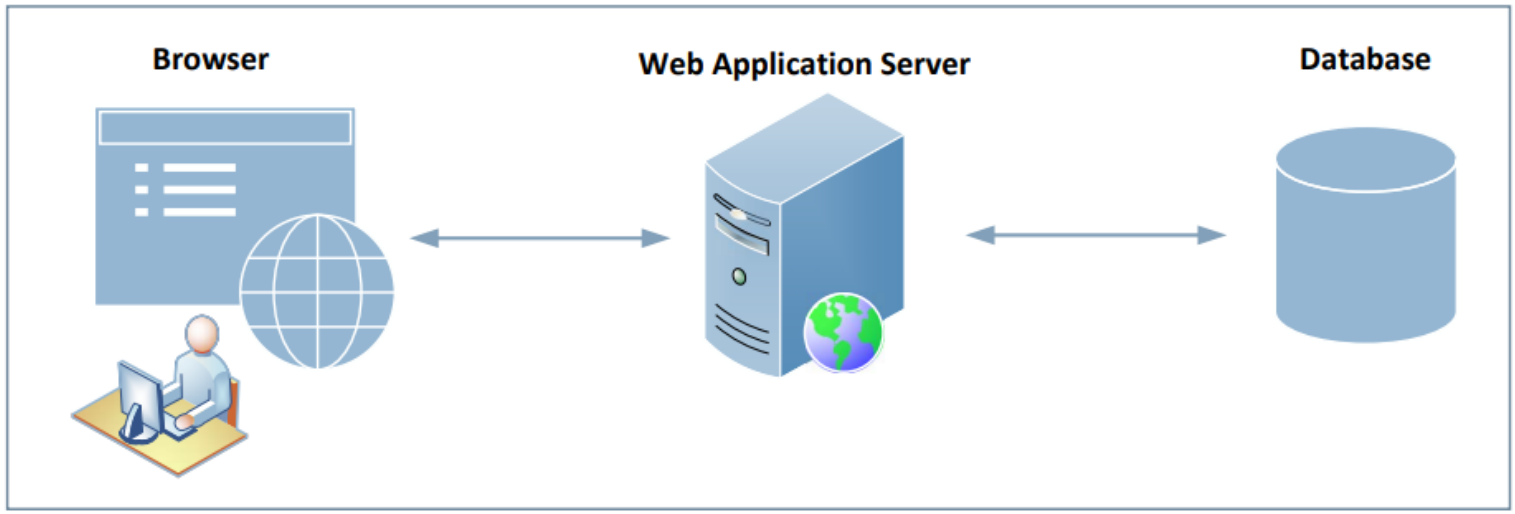
WEB SECURITY BASICS

CS-5156/CS-6056: SECURITY VULNERABILITY ASSESSMENT (SPRING 2025)
LECTURE 13

Outline

- The web architecture
- Web server
- HTTP protocol, cookies
- JavaScript and sandboxing

The Web Architecture



HTML

- Hypertext Markup Language
- For creating web pages
- Example

```
<html>
<body>
   <h1>Heading</h1>
   <p>This is a test.</p>
</body>
</html>
```

bank32/index.html

CSS: Cascading Style Sheets

- Specify the presentation style
- Separate content from the presentation style
- Example

```
<style type="text/css">
  .myclass   { background-color: yellow; }
  #myid { position:absolute; top:220px; left:700px; }
  body  { background-color: lightblue;
          margin-top:    50px; margin-bottom: 20px;
          margin-right:   0px; margin-left:   80px; }
  h1    { font-family: Arial, Helvetica, sans-serif; }
</style>
```

bank32/css-example.html

Dynamic Content

- Adobe Flash (and ActionScript)
  - Obsolete (replaced by HTML5)
  - Today: HTML5 compliant browsers can run any multimedia content without relying on third-party plugins.
- Microsoft Silverlight
  - Obsolete (replaced by HTML5) - deprecated in **2012**
- ActiveX (Microsoft)
  - Many security issues and lack of cross-platform support
  - Deprecated in **2015** (Not included with Edge browser)
- Java applets
  - Obsolete (not supported by android and iOS mobile devices)
- **JavaScript**
  - **Most popular and dominant**

JavaScript

- Also known as **ECMAScript**
- Scripting language for web pages
- Three different ways to include JavaScript code

1
```
<script>
  ... Code ...
</script>
```

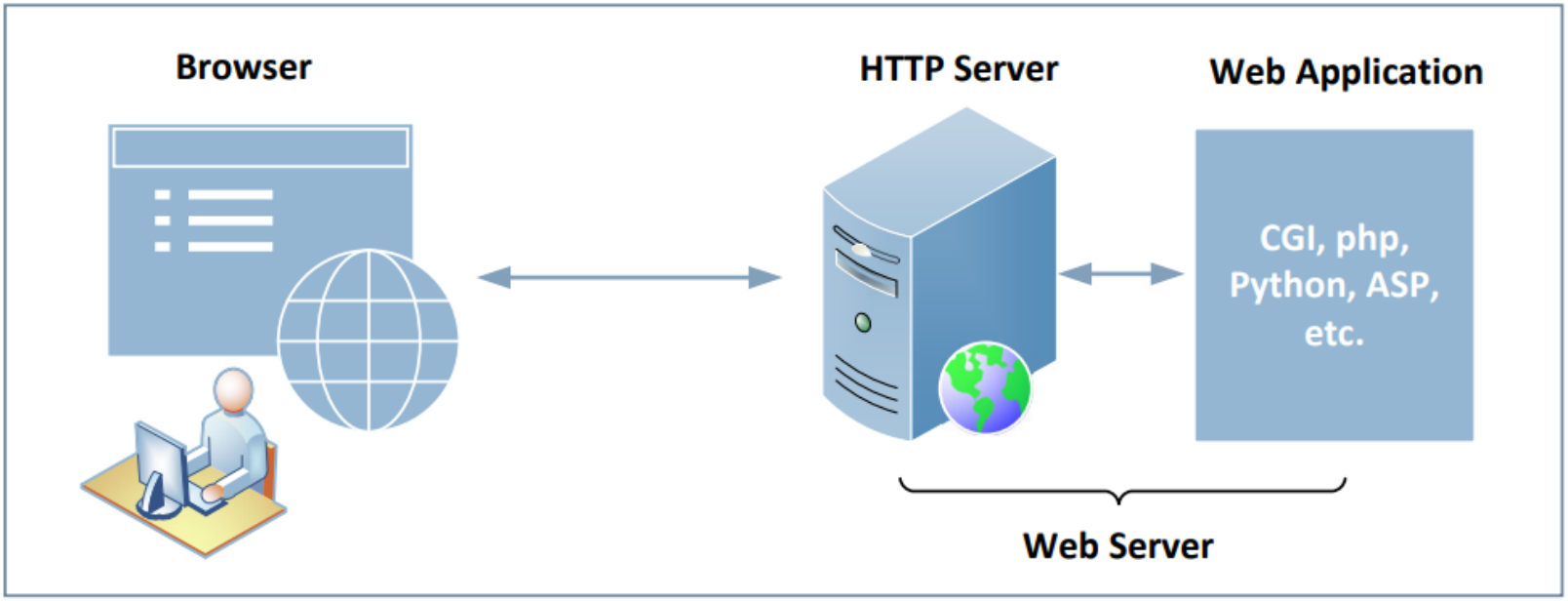
2
```
<script src="myScript.js"></script>
<script src="https://www.example.com/myScript.js"></script>
```

3
```
<button type="button" onclick="myFunction()">Click it</button>
```

bank32/js-example.html

WEB SERVER

HTTP Server & Web Application Server



Case Study: **Apache** Server

Configuration: Virtual Hosting "**my_server.conf**"

```
<VirtualHost *:80>
    ServerName www.bank32.com
    DocumentRoot "/var/www/bank32"
</VirtualHost>

<VirtualHost *:80>
    ServerName www.bank99.com
    DocumentRoot "/var/www/bank99"
</VirtualHost>
```

Demo Setup

```
seed@VM:~/.../lecture13$ docker-compose build
Building apache-server
Step 1/3 : FROM handsonsecurity/seed-server:apache-php
 ---> 2365d0ed3ad9
Step 2/3 : COPY my_server.conf server_name.conf /etc/apache2/sites-available/
 ---> Using cache
 ---> 8f8416f729e4
Step 3/3 : RUN  a2ensite server_name.conf        && a2ensite my_server.conf
 ---> Using cache
 ---> 266340050559

Successfully built 266340050559
Successfully tagged seed-image-apache-server:latest
seed@VM:~/.../lecture13$
```

Demo Setup

```
seed@VM:~/.../lecture13$ docker-compose up
Creating apache-10.9.0.5 ... done
Attaching to apache-10.9.0.5
apache-10.9.0.5  |  * Starting Apache httpd web server apache2                    *
```

Demo Setup

If docker container failed to start, you may try to stop and/or remove all existing containers

```
seed@VM:~/.../lecture13$ docker stop $(docker ps -aq)
1169073d069b
1d5e4c22697d
021b5f5c4c09
1af138398b25
d46dc0784b56
seed@VM:~/.../lecture13$ docker rm $(docker ps -aq)
1169073d069b
1d5e4c22697d
021b5f5c4c09
1af138398b25
d46dc0784b56
```

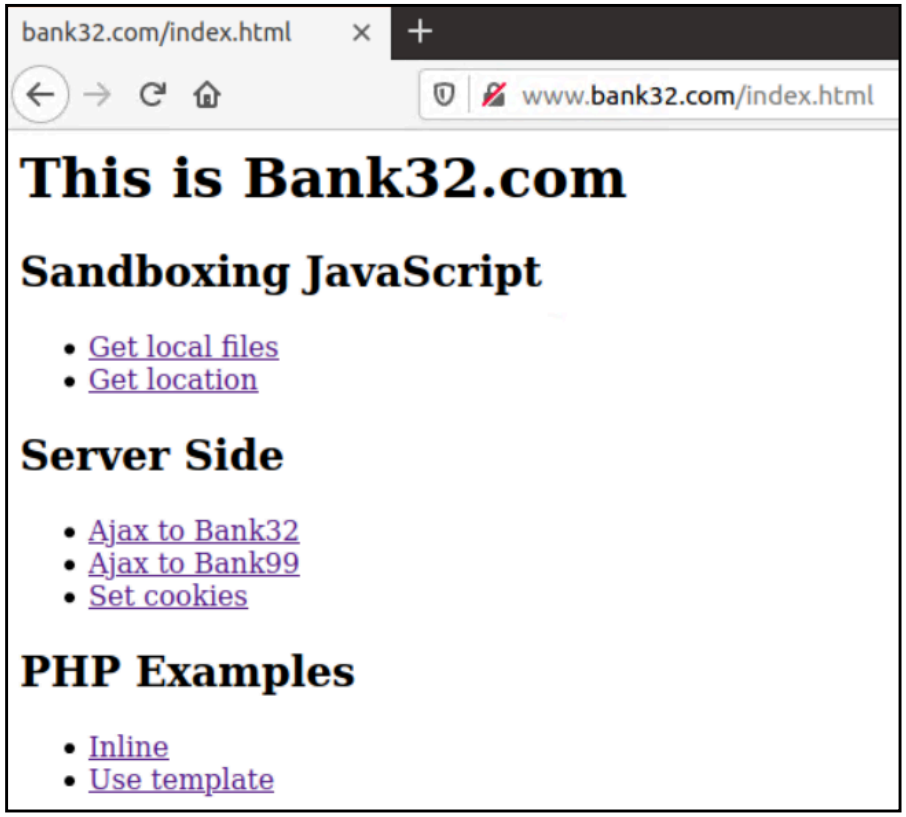
Demo Setup

- Update /etc/hosts file to include the following lines

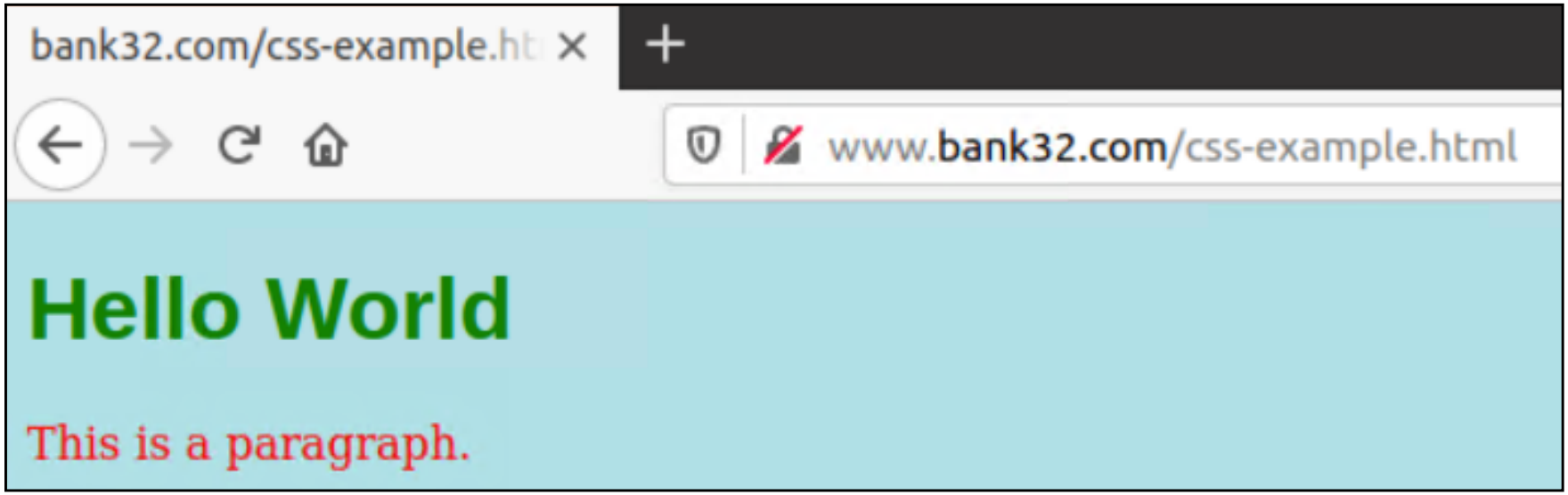
```
[04/07/23]seed@VM:/$ sudo vim /etc/hosts
```

```
# For Web Security Basics
10.9.0.5        www.bank32.com
10.9.0.5        www.bank99.com
```

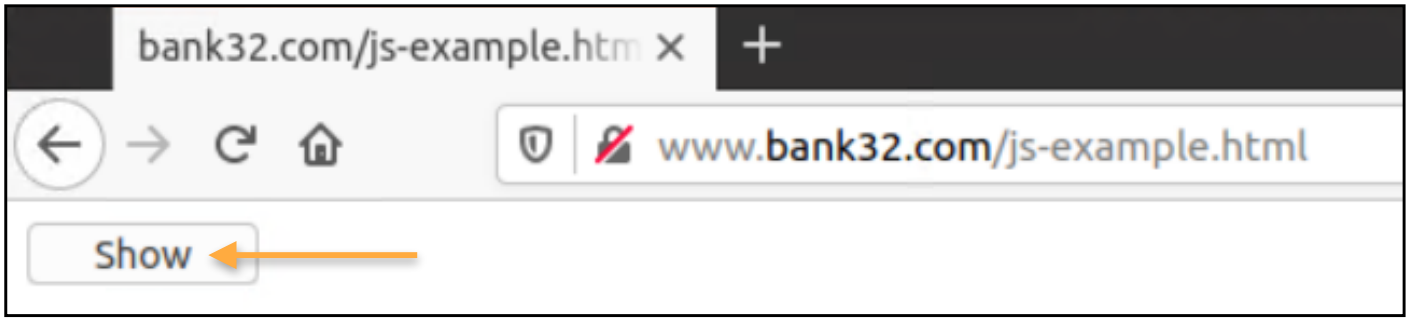
HTML Example

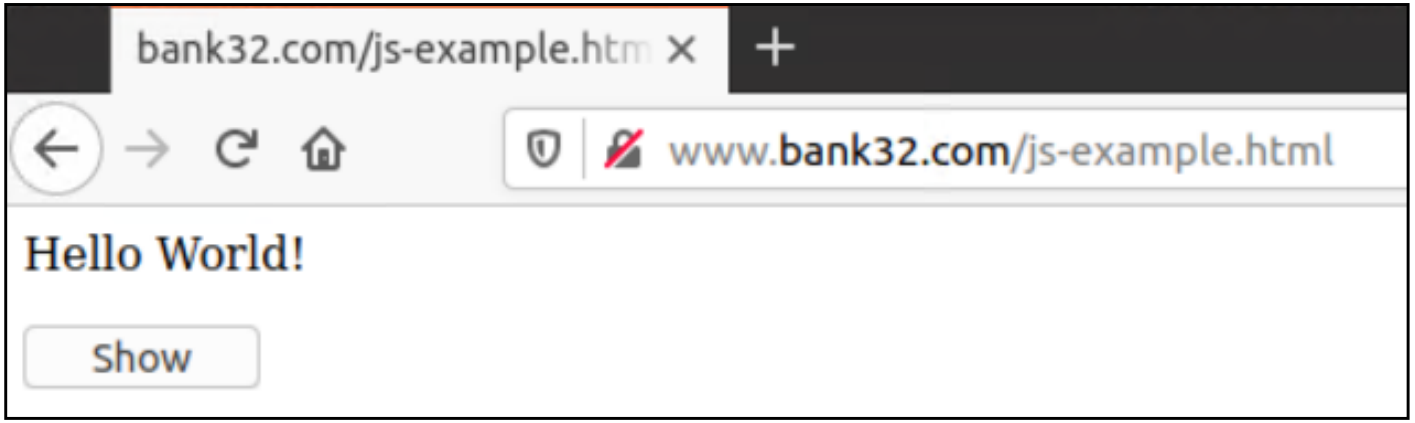


CSS Example


bank32.com/css-example.ht
www.bank32.com/css-example.html
Hello World
This is a paragraph.

JS Example


bank32.com/js-example.htm
www.bank32.com/js-example.html
Show


bank32.com/js-example.htm
www.bank32.com/js-example.html
Hello World!
Show

How HTTP Server Interacts with Web Applications

- **CGI: The Common Gateway Interface**
  - Starts the CGI program in a **new process**
    - **Connects to standard input/output using a pipe**
  - Each process serves one HTTP request
    - The request information is passed via Environment Variables
    - The data (request/response) is passed via the pipe
- FastCGI: a variation of the CGI, faster
  - One **persistent process** that handles multiple requests
- **Modules: directly execute script-based programs**
  - **E.g., /etc/apache2/mods-available (php, perl, ruby, etc)**

PHP Example

Inline Approach

```
<!doctype html>
<html>
<body>
<h1>PHP Experiment</h1>
<h2>Current time is
<?php echo date("Y-m-d h:i:sa") ?>
</h2>
</body>
</html>
```

Template Approach

```
<?php
 $title = "PHP Experiment";
 $time = date("Y-m-d h:i:sa")
?>

<!doctype html>
<html>
<body>
<h1><?=$title?></h1>
<h2>Current time is <?=$time?></h2>
</body>
</html>
```

PHP Example

Inline Approach

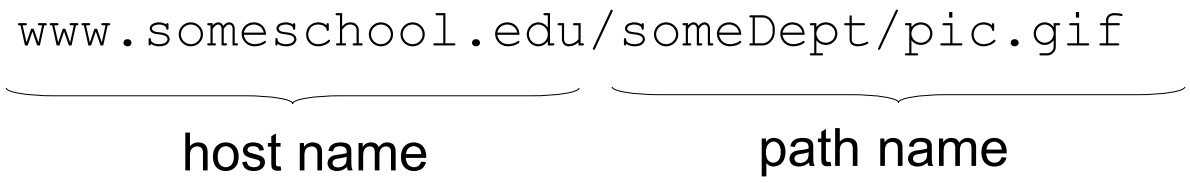
Template Approach

HTTP PROTOCOL

Web and HTTP

First, a quick review…

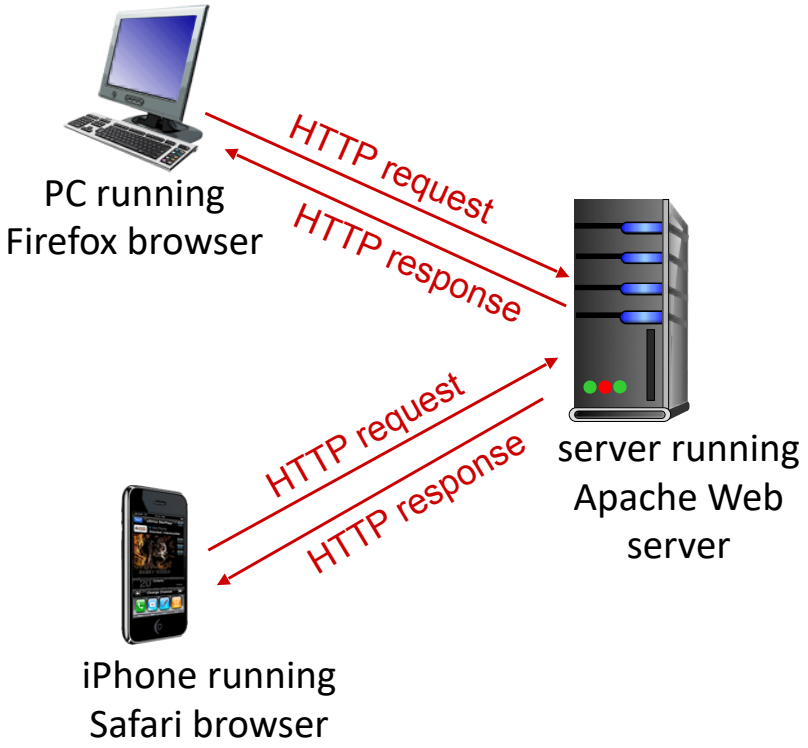
- web page consists of *objects,* each of which can be stored on different Web servers
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects, each* addressable by a *URL,* e.g.,



HTTP overview

HTTP: hypertext transfer protocol

- Web's application-layer protocol
- client/server model:
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests



HTTP overview (continued)

HTTP uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is "stateless"

- server maintains *no* information about past client requests

note

protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

HTTP connections: two types

Non-persistent HTTP

1. TCP connection opened
2. at most one object sent over TCP connection
3. TCP connection closed

downloading multiple objects required multiple connections

Persistent HTTP

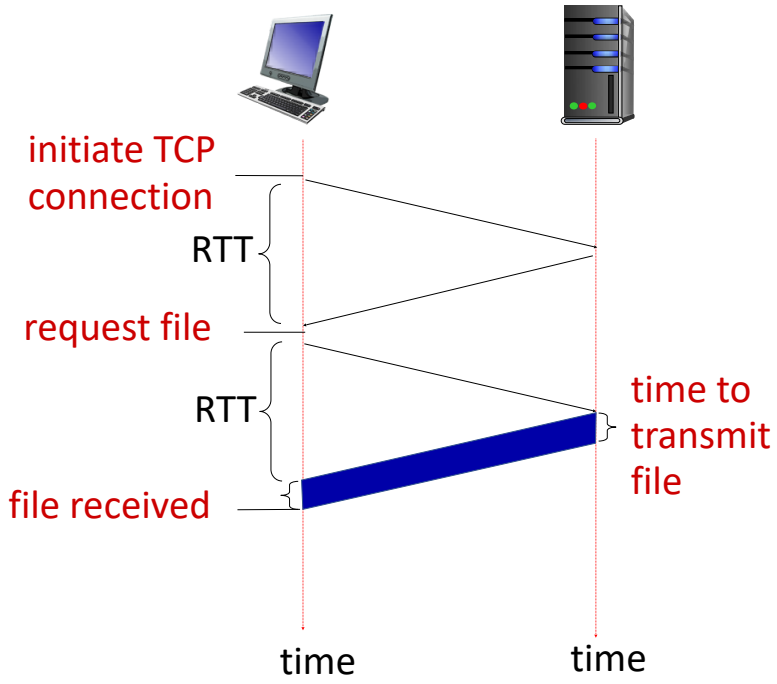
- TCP connection opened to a server
- multiple objects can be sent over ***single*** **TCP connection** between client, and that server
- TCP connection closed

Non-persistent HTTP: response time

RTT (definition): time for a small packet to travel from client to server and back

HTTP response time (per object):

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- object/file transmission time



Non-persistent HTTP response time = 2RTT+ file transmission time

Persistent HTTP (HTTP 1.1)

Non-persistent HTTP issues:

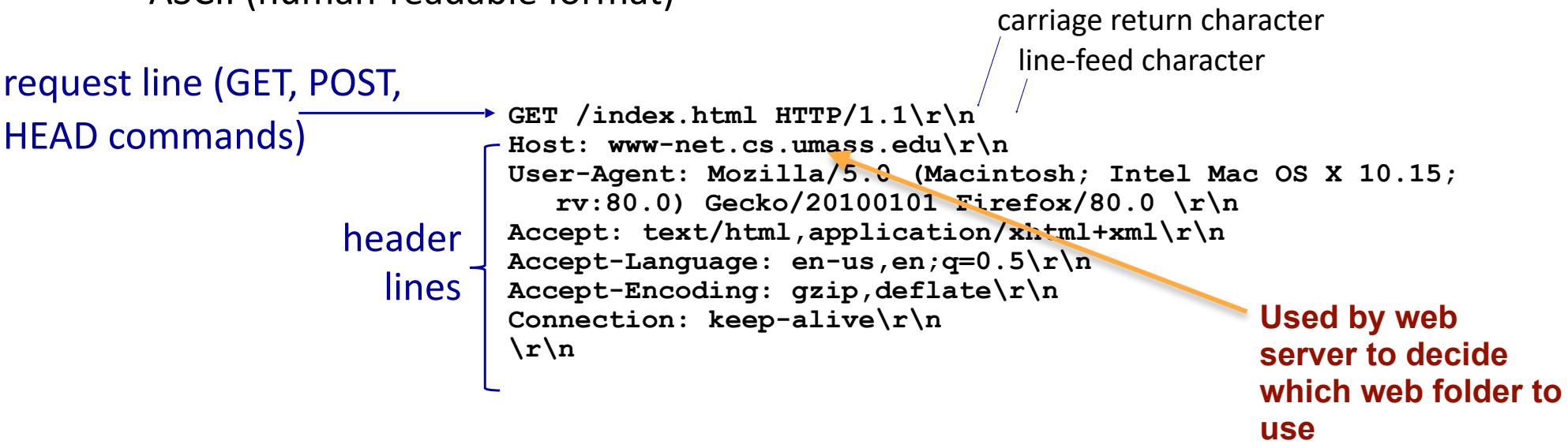
- **requires 2 RTTs per object**
- OS overhead for *each* TCP connection
- browsers often open multiple parallel TCP connections to fetch referenced objects in parallel
  - Overwhelming server

Persistent HTTP (HTTP1.1):

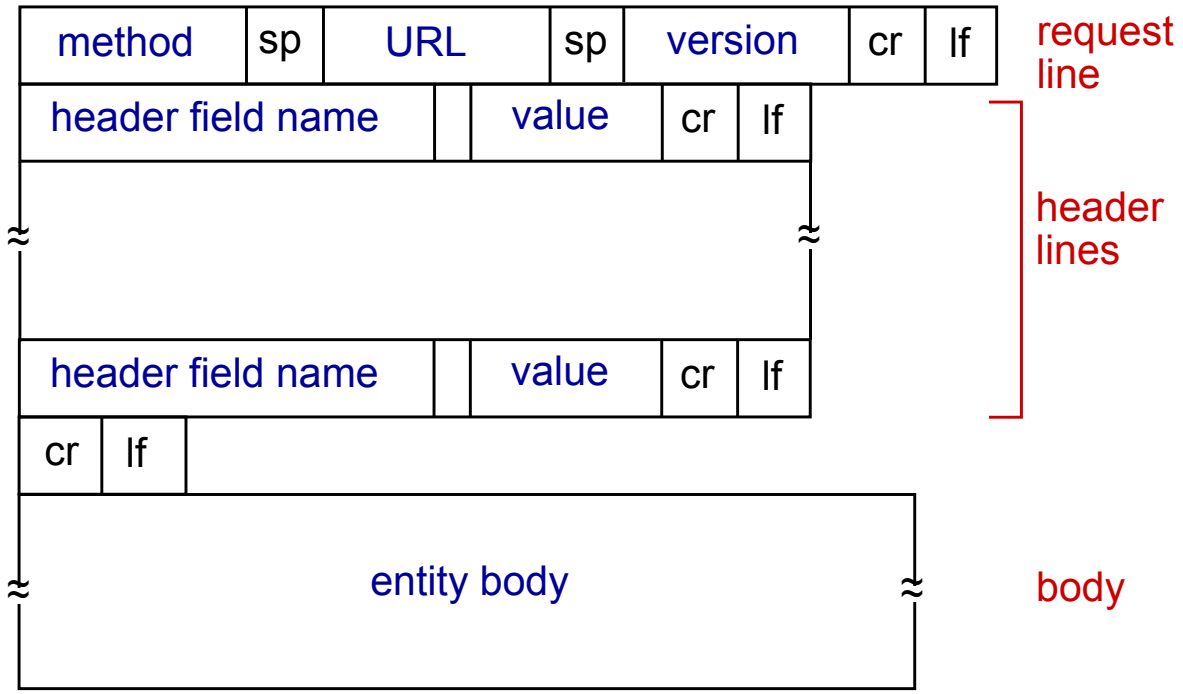
- server leaves connection open after sending response
- **subsequent HTTP messages between same client/server sent over open connection**
- client sends requests as soon as it encounters a referenced object
- **as little as one RTT for all the referenced objects** (cutting response time in half) - no new connection establishment needed

HTTP request message (example)

- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)



HTTP request message: general format



Other HTTP request messages

POST method:

- web page often includes form input
- user input sent from client to server in entity body of HTTP POST request message

GET method (for sending data to server):

- include user data in URL field of HTTP GET request message (following a '?'):

```
www.somesite.com/animalsearch?monkeys&banana
```

HEAD method:

- requests headers (**only**) that would be returned *if* specified URL were requested with an HTTP GET method.
- Same as GET method but response does not return anything in the body

PUT method:

- uploads new file (object) to server
- completely replaces file that exists at specified URL with content in entity body of POST HTTP request message

GET versus POST Requests

- Main difference
  - how they send data to the server

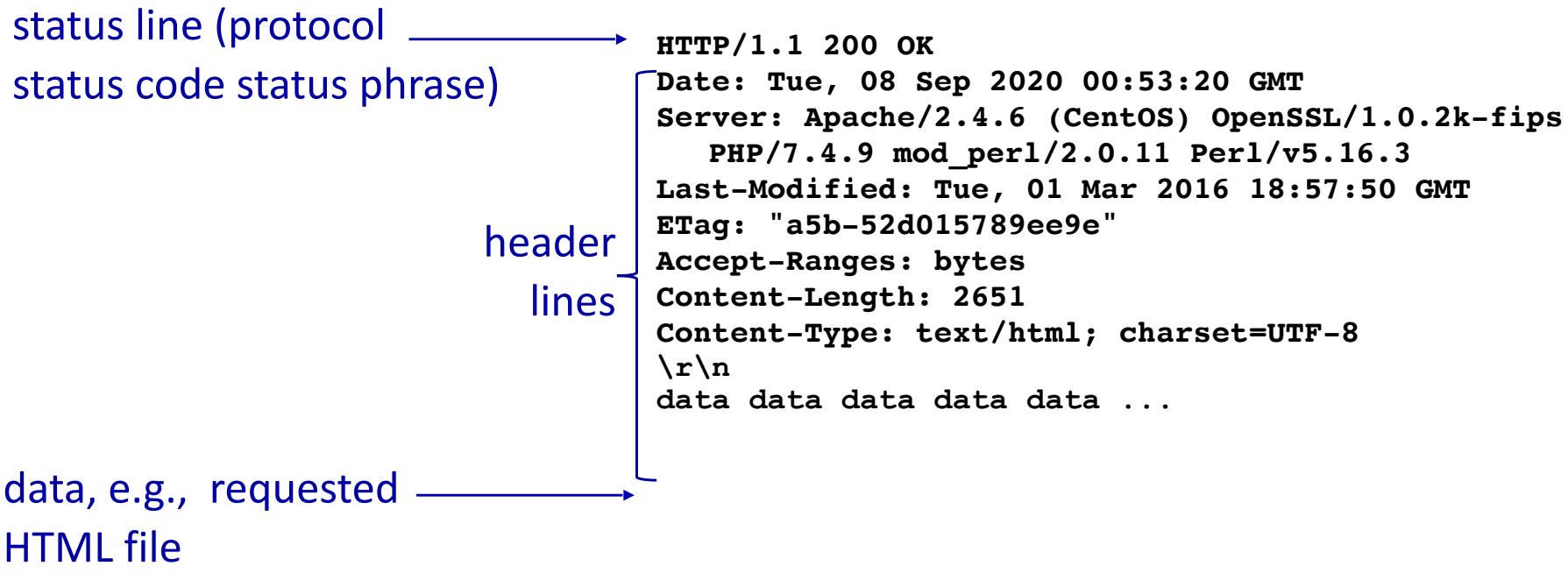
- **GET** request
  - Privacy concerns

```
GET /post_form.php?foo=hello&bar=world HTTP/1.1
Host: www.example.com
Cookie: SID=xsdfgergbghedvrbeadv
```

- **Post** request

```
POST /post_form.php HTTP/1.1
Host: www.example.com
Cookie: SID=xsdfgergbghedvrbeadv
Content-Length: 19
foo=hello&bar=world
```

HTTP response message



HTTP response status codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:

200 OK

- request succeeded, requested object later in this message

301 Moved Permanently

- requested object moved, new location specified later in this message (in **Location: field**)

400 Bad Request

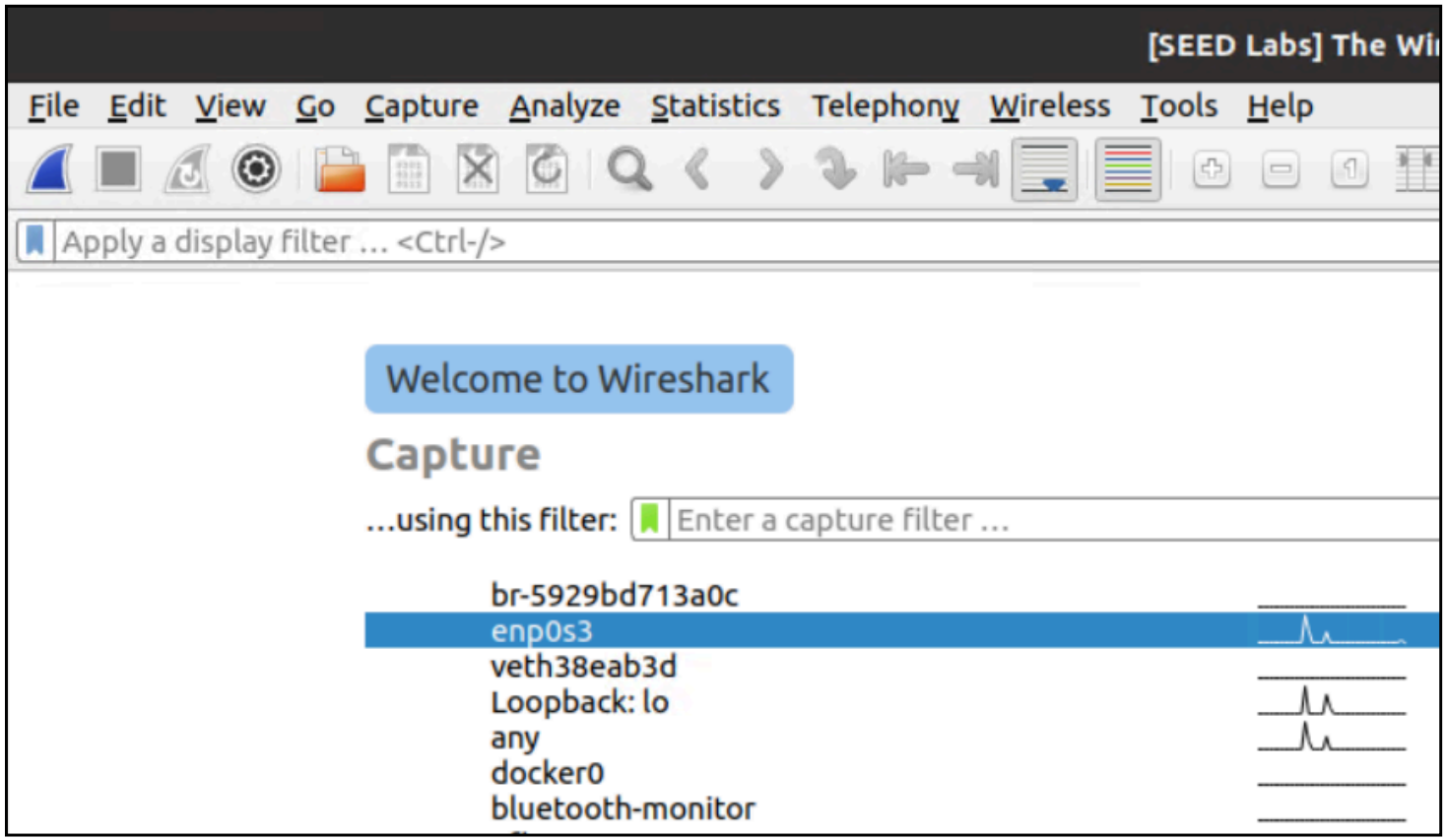
- request msg not understood by server

404 Not Found

- requested document not found on this server

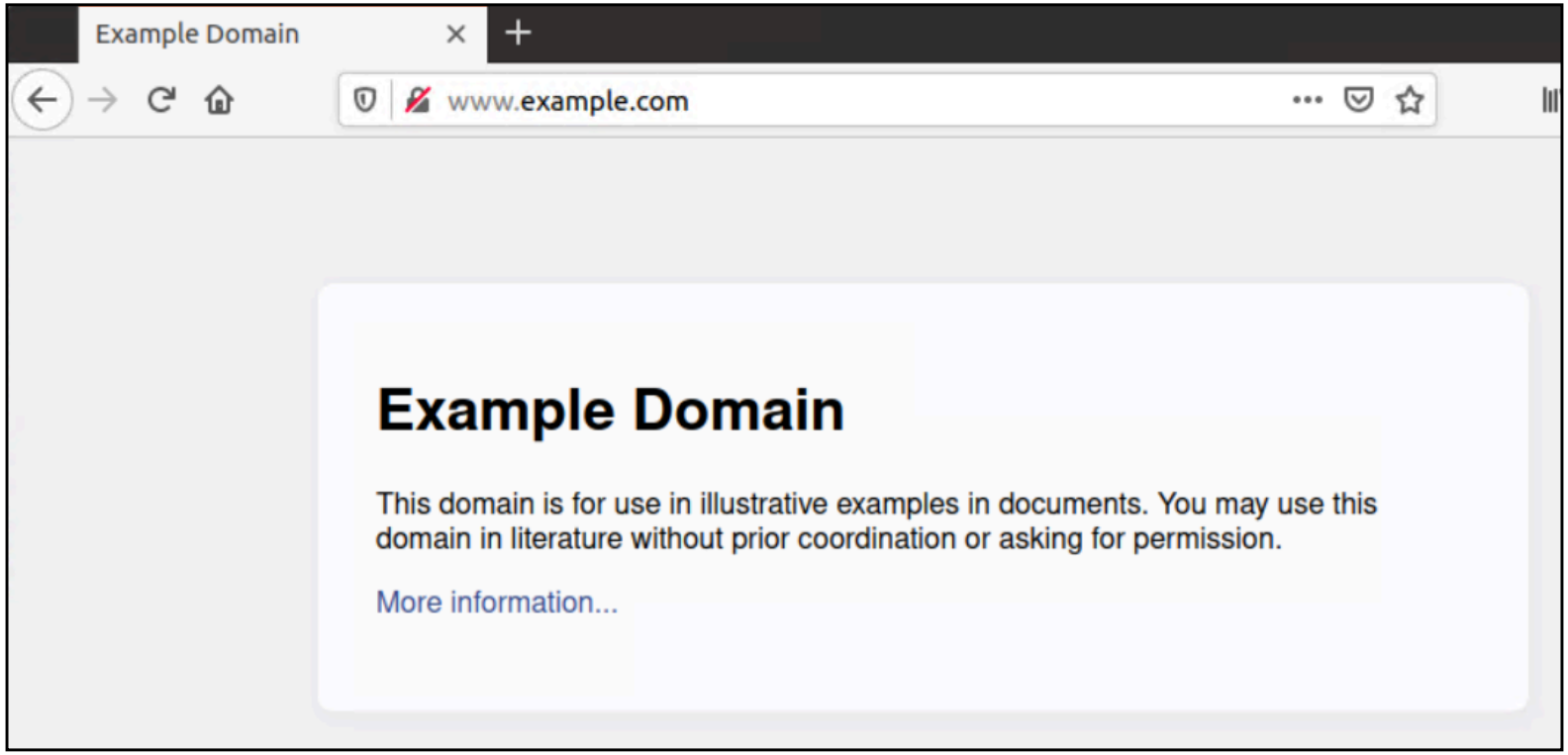
505 HTTP Version Not Supported

Example HTTP Response

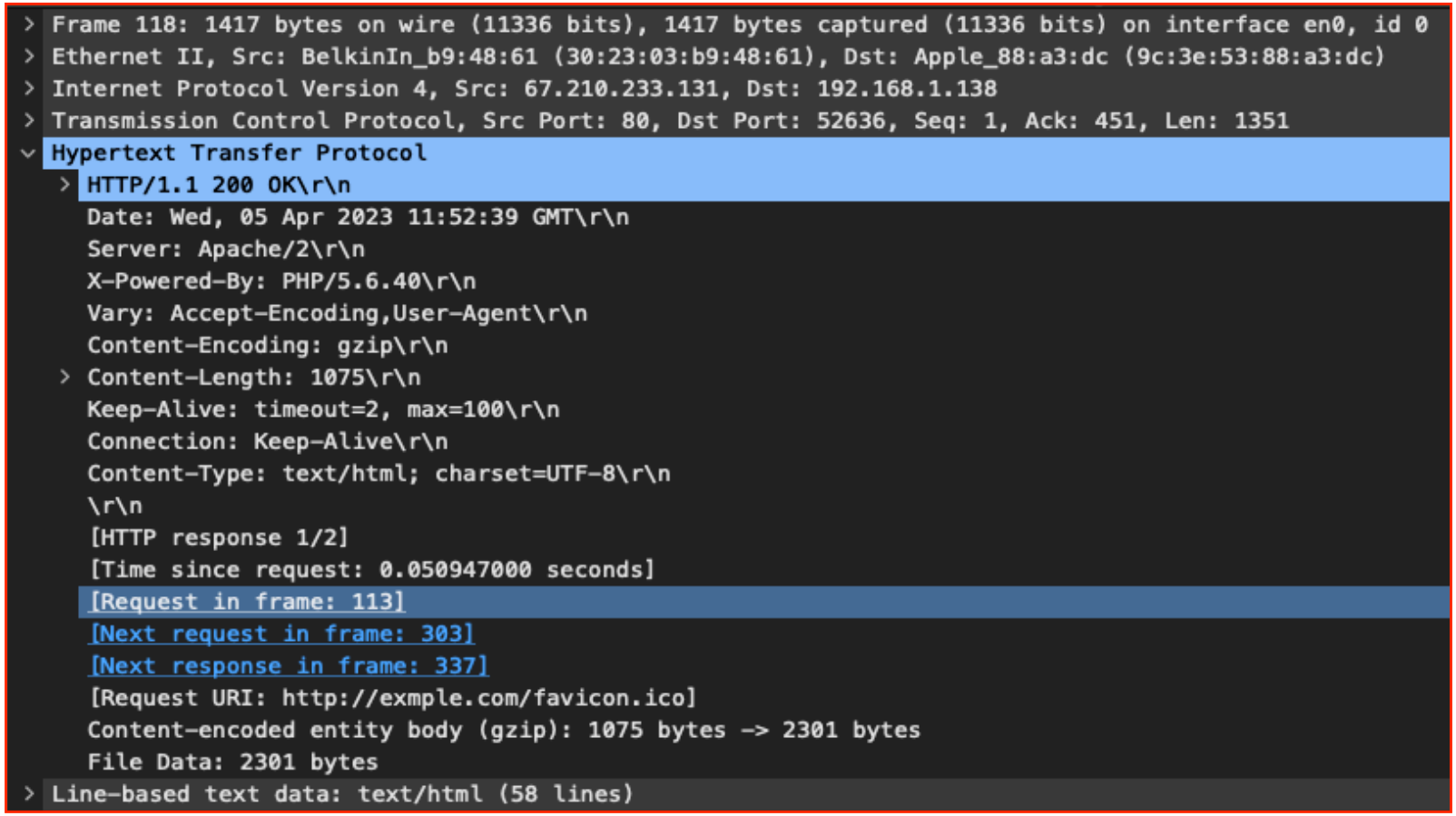


Example HTTP Response

- Restart Browser first



Example HTTP Response



Example HTTP Response

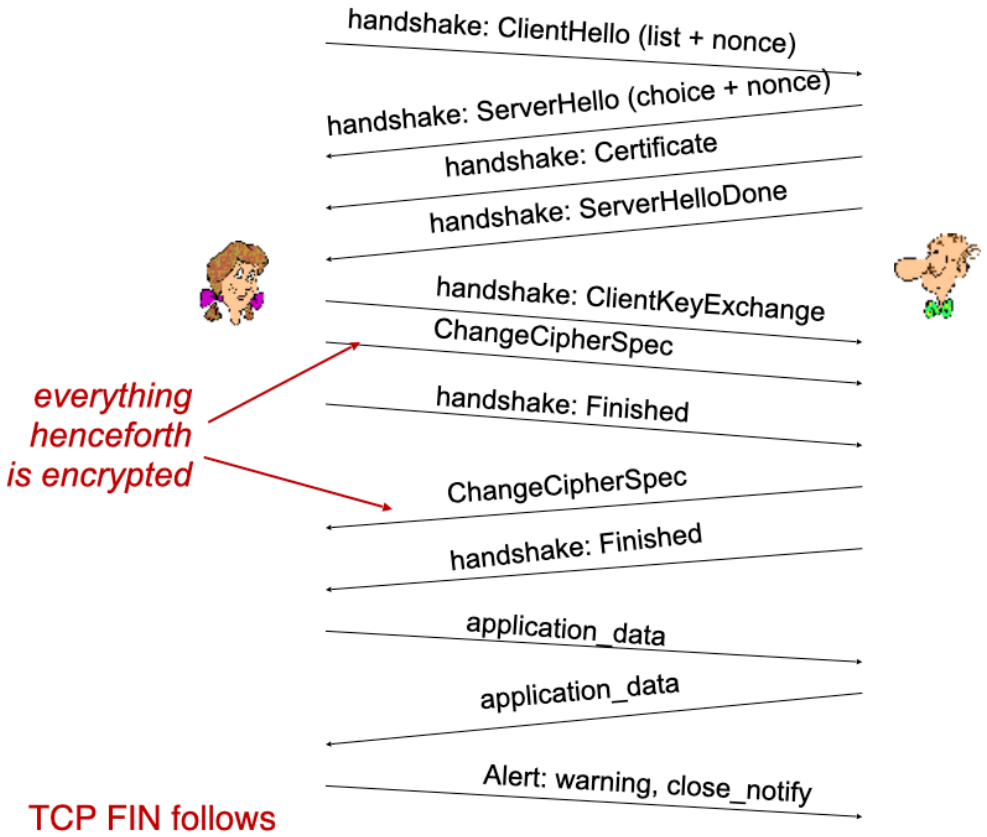
```
Line-based text data: text/html (58 lines)
    <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">\n
    <html>\n
    <head>\n
    <title>Example Website, Exmple Website - Exmple.com</title>\n
      <meta name="description" content="This is an example website that can be used as a demo when creating a dumm
      <meta name="keywords" content="Exmple,Example,Exmple.com">\n
    </head>\n
    \n
    <body>\n
    \n
    <center><table style="margin-top: 2px" width=640><tr><td>\n
    \n
    <h2>Example Website</h2>\n
    \n
    <p>This is an example website that can be used to demonstrate links without really linking\n
        to any real site. The domains example.com, example.net and example.org are reserved\n
        and cannot be followed through a link, while this misspelled domain exmple.com work\n
        as intended when linked to.\n
        \n
    \n
    <p>The main page if this example website is located at <a href="http://www.exmple.com">Exmple, Example</a>\n
        and is not much different from this page\n
    \n
        \n
    <p>The following links are examples of websites on the internet:\n
```

HTTPS

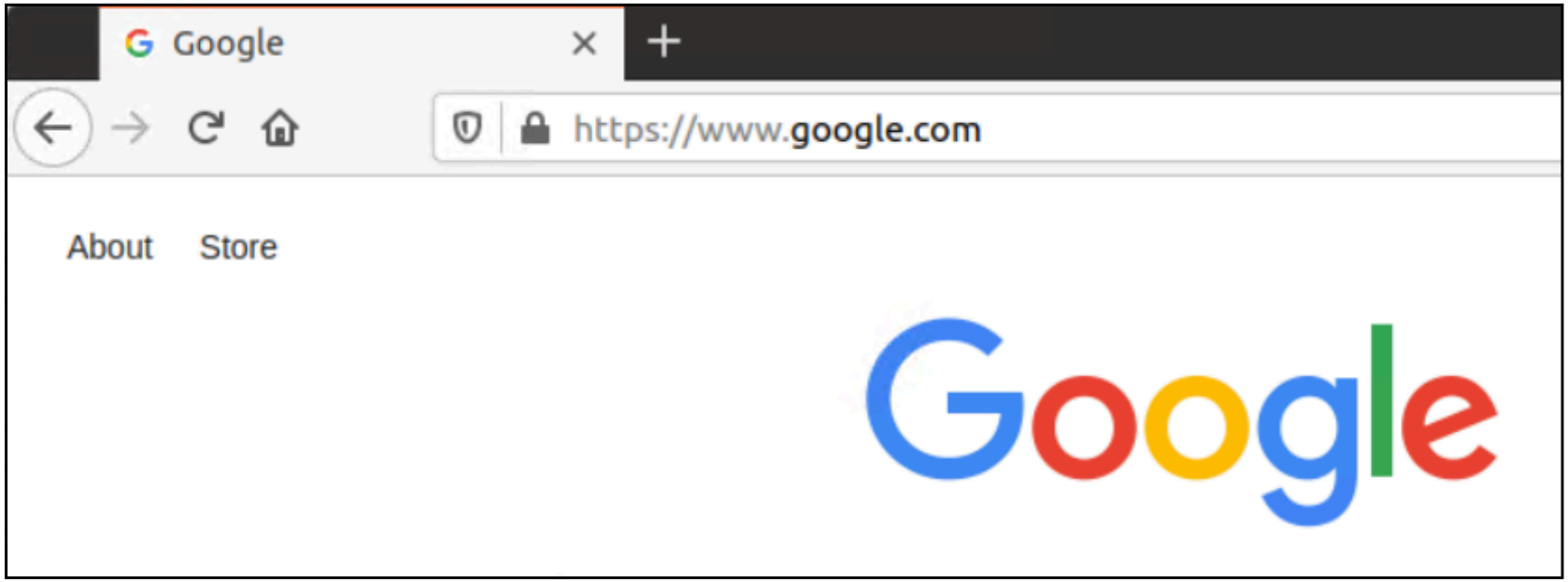
- HTTP over TLS/SSL

SSL

1. client sends list of algorithms it supports, along with client **nonce**
2. server chooses algorithms from list; sends back: **choice** + **certificate** + server **nonce**
3. client verifies certificate (with CA), extracts server's public key, generates pre_master_secret (PMS), encrypts PMS with server's public key, sends encrypted PMS to server
4. **client and server** independently **compute** the four encryption and MAC keys (for authentication) from PMS and nonces
5. client sends a MAC of all the handshake messages
6. server sends a MAC of all the handshake messages



Example TLS Capture

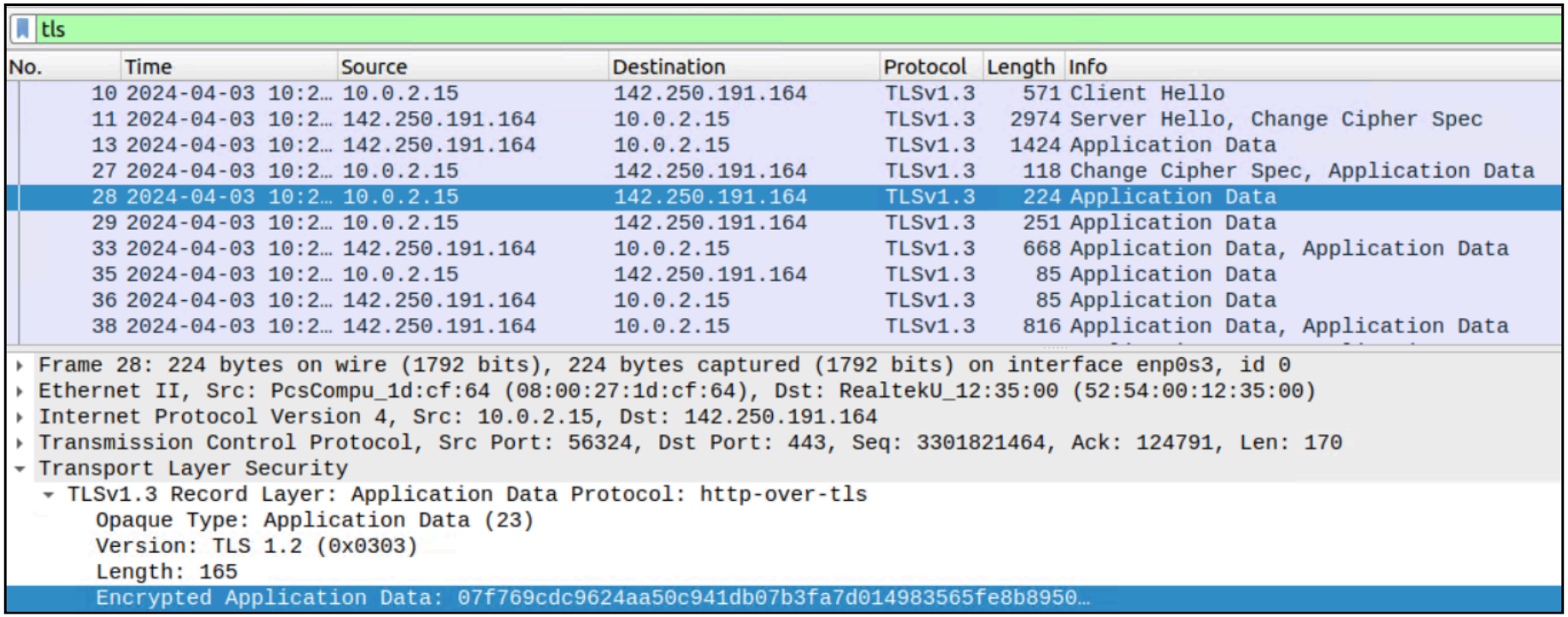


Example TLS Handshake Messages

tls

No.	Time	Source	Destination	Protocol	Length	Info
10	2024-04-03 10:2…	10.0.2.15	142.250.191.164	TLSv1.3	571	Client Hello
11	2024-04-03 10:2…	142.250.191.164	10.0.2.15	TLSv1.3	2974	Server Hello, Change Cipher Spec
13	2024-04-03 10:2…	142.250.191.164	10.0.2.15	TLSv1.3	1424	Application Data
27	2024-04-03 10:2…	10.0.2.15	142.250.191.164	TLSv1.3	118	Change Cipher Spec, Application Data

Example TLS Response



Cookies

- Web server is stateless
  - does not maintain a long-term connection with the client
- HTTP Cookies: used to save information on the client side
  - Browser save cookies
  - Attach cookies in every request

Maintaining user/server state: cookies

Web sites and client browser use *cookies* to maintain some state between transactions

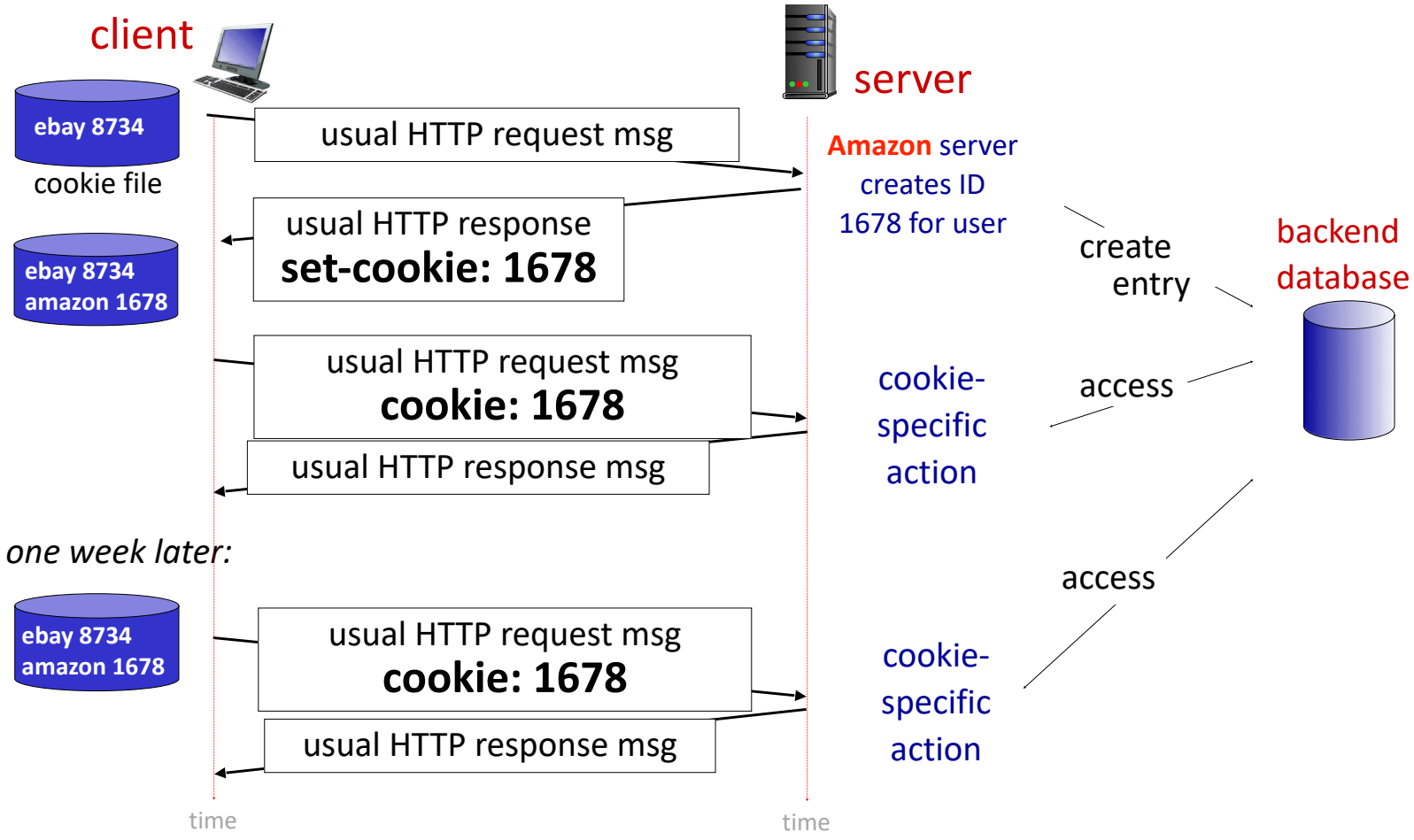
four components:

1) **cookie header line** of HTTP ***response*** message
2) **cookie header line** in **next** HTTP ***request*** message
3) **cookie file** kept on user's host, managed by user's browser
4) **back-end database** at Web site

Example:

- Susan uses browser on laptop, visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID (aka "cookie")
  - entry in backend database for ID
- subsequent HTTP requests from Susan to this site will contain cookie ID value, allowing site to "identify" Susan

Maintaining user/server state: cookies



HTTP cookies: comments

What cookies can be used for:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

Challenge: How to keep state?

- *at protocol endpoints:* maintain state at sender/receiver over multiple transactions
- *in messages:* cookies in HTTP messages carry state

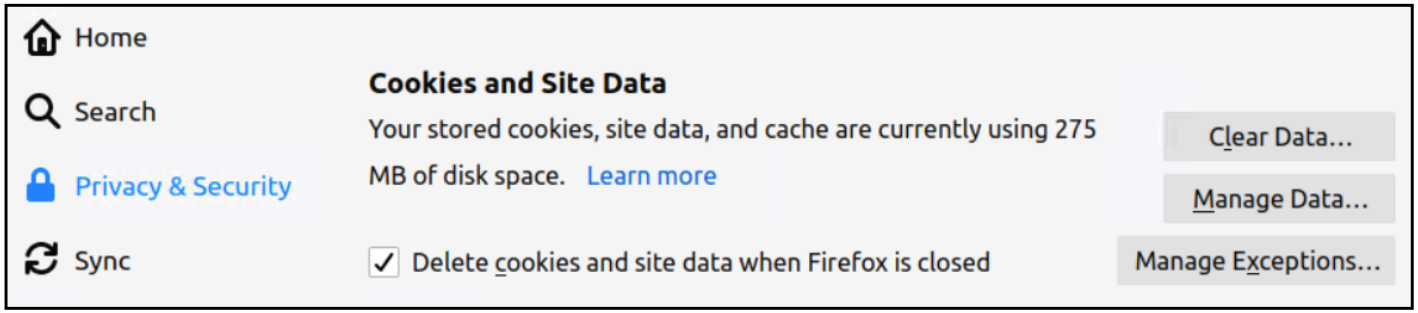
Privacy

cookies and privacy:

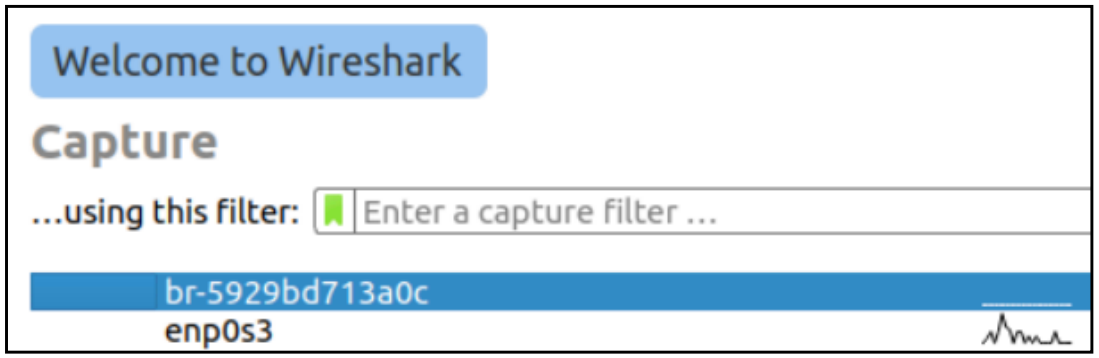
- cookies permit sites to *learn* a lot about you on their site.
- third party persistent cookies (tracking cookies) allow common identity (cookie value) to be tracked across multiple web sites

Setting Cookies (Experiment Setup)

- Clearing browser cookies and restart browser



- Wireshark capture interface



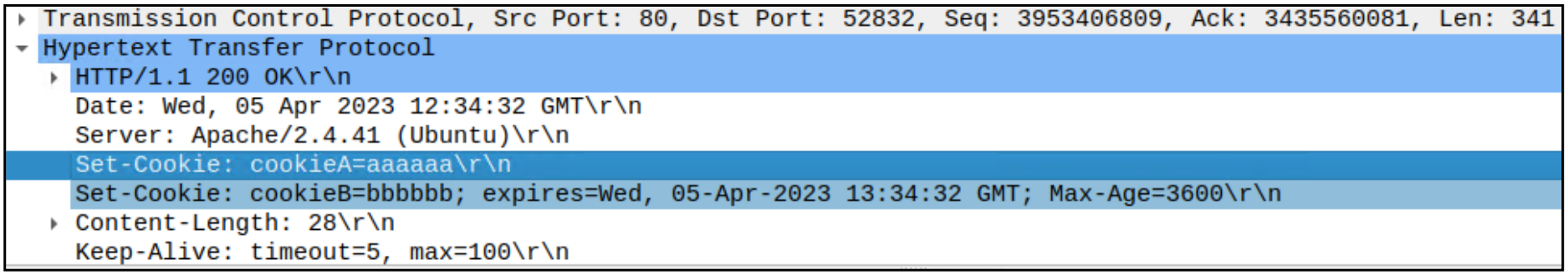
Setting Cookies

- Server: setting cookies

```php
<?php
  setcookie('cookieA', 'aaaaaa');
  setcookie('cookieB', 'bbbbbb', time() + 3600);

  echo "<h2>Cookies are set</h2>"
?>
```

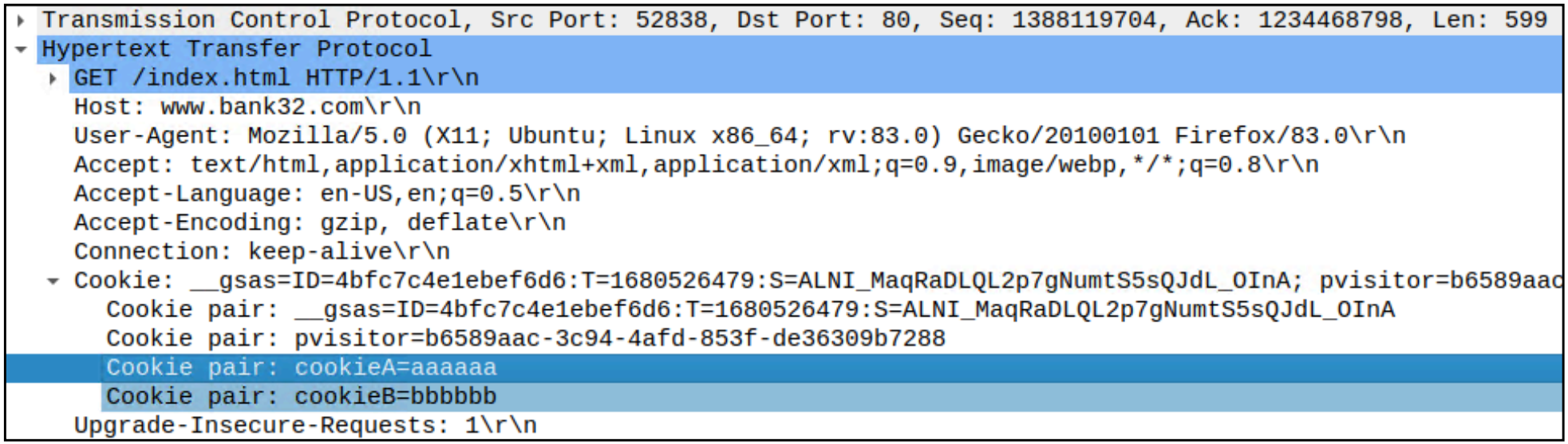
bank32/setcookies.php

- HTTP response



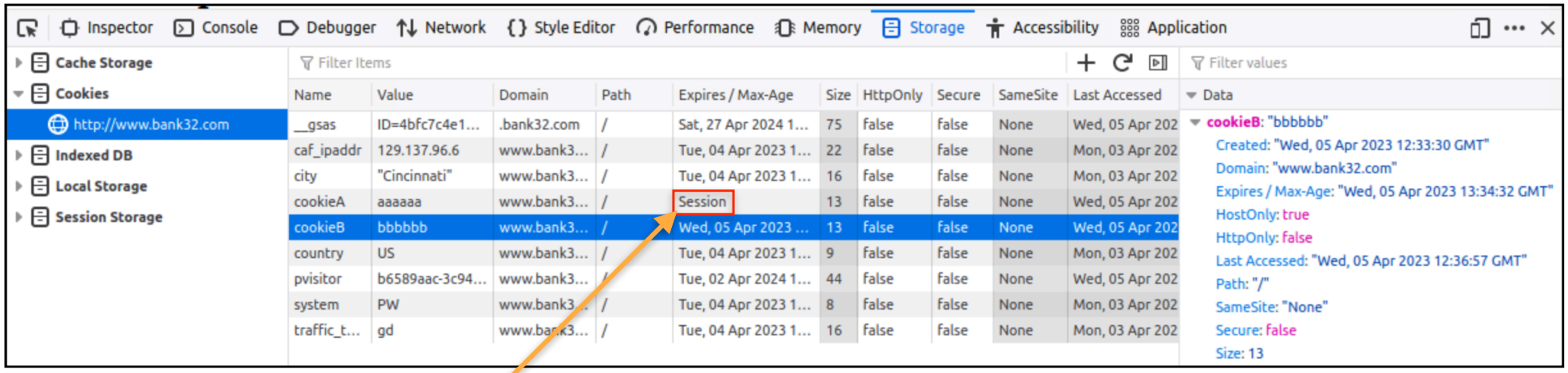
Attaching Cookies

- Browser: attach all the cookies belonging to the target server



Cookie File

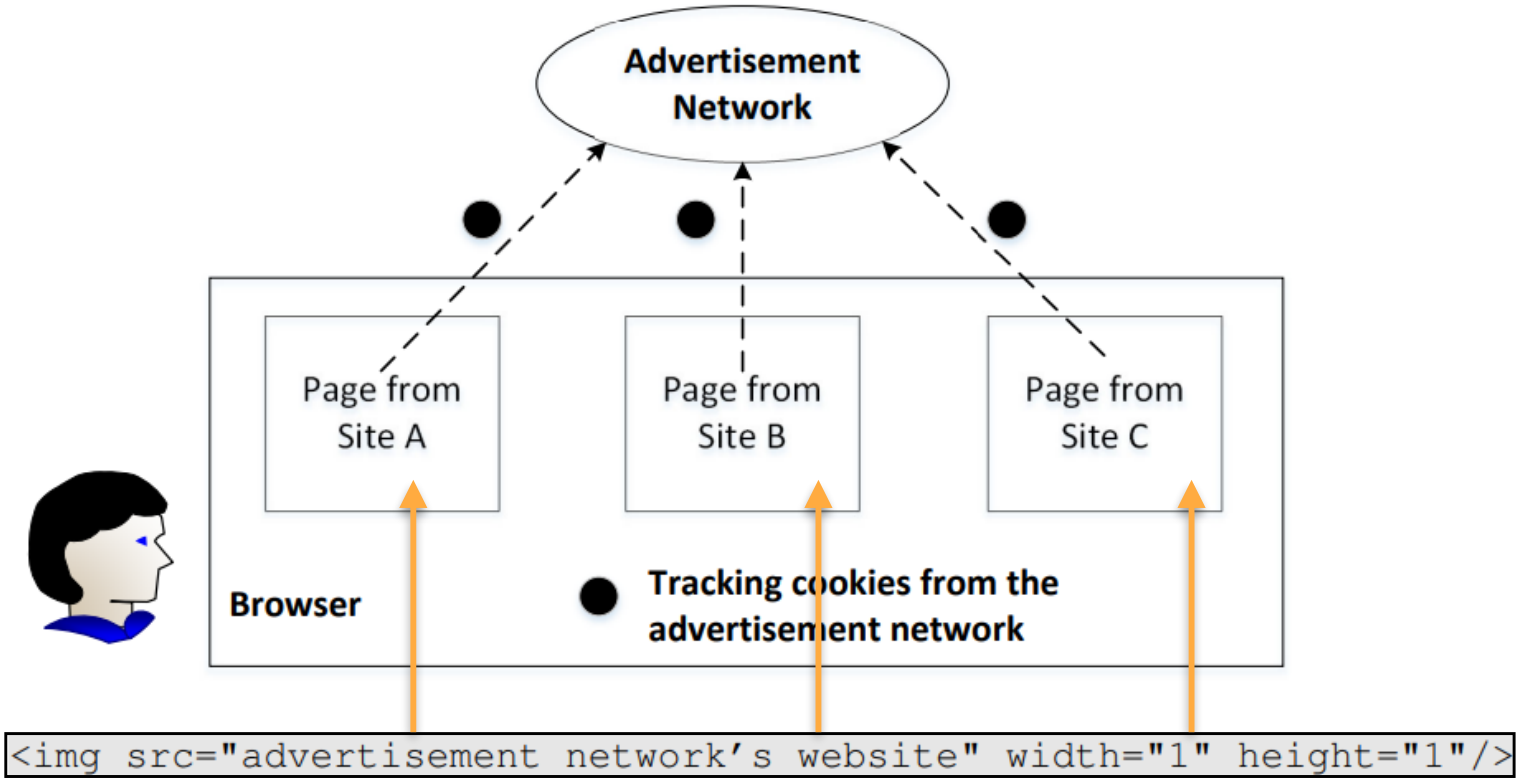
- Accessing Browser's Cookies File / Storage



Typically expires at end of session (when browser is closed).

Vague meaning: depending on OS and browser, can expire after user closes tab, closes the browser, or never expire even after closing the browser

Tracking Using Cookies



Same tag in html code of all 3 sites visited

Prevent Tracking

- Using **anonymous mode** in browsing (e.g., Chrome Incognito mode, Private browsing in Safari and Firefox)
  - Browsing history and cookies are deleted after closing the browser window of exiting the anonymous mode

- **Block third-party cookies**
  - First-party cookies are essential for browsing
  - Third-part cookies are mainly used for advertisement, information collection, etc.
  - E.g., By default, Safari blocks all third-party cookies (user can change settings to allow third-party cookies)

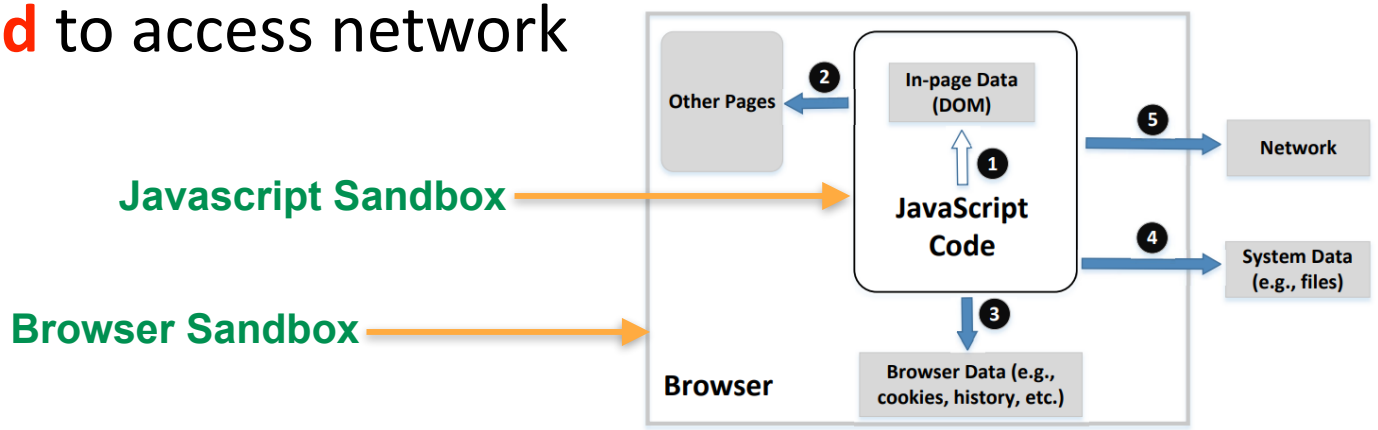
Session Cookies

- A cookie: store **session ID**
- The session ID identifies a session
- Session data are typically maintained on the server
- Session is typically created after user login
  - **Have the session ID = have the access**
  - Session can expire if some checks did not pass
    - e.g., password change (session is marked invalid on server when user changes password)
  - Security sensitive
  - ID: Random number (e.g., a **128-bit UUID**)

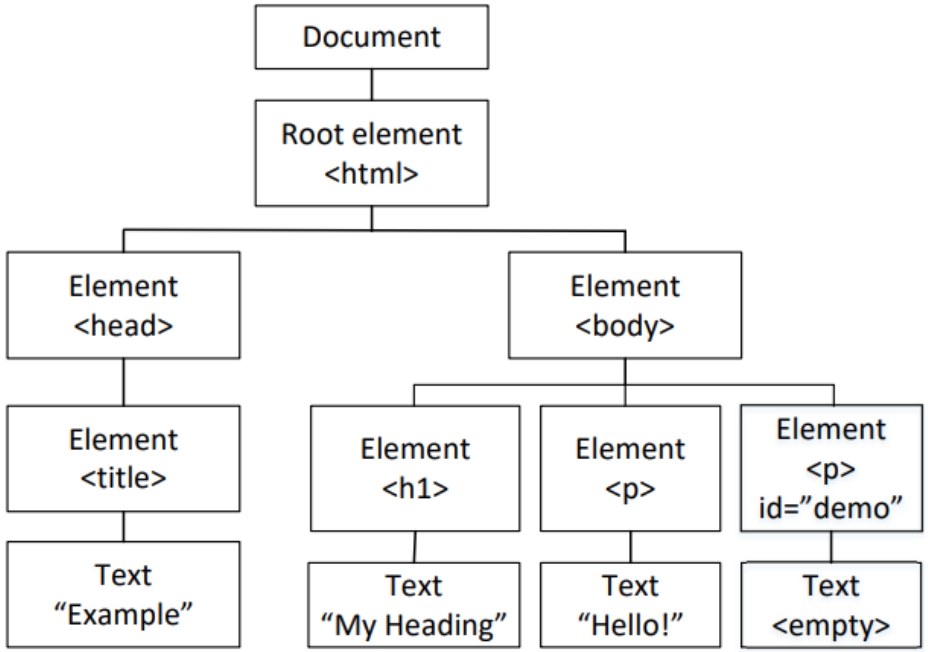
JAVASCRIPT AND SANDBOX

Protection Needs

- Javascript code should
  1) be **allowed** to access data on same page (**DOM**)
  2) **not** be **allowed** to access data inside **another page**
  3) be able to access browser data but in a **controlled manner**
  4) not be allowed to access user's file system **without user's consent**
  5) **not** be **allowed** to access network



1) Access In-Page Data using **DOM**

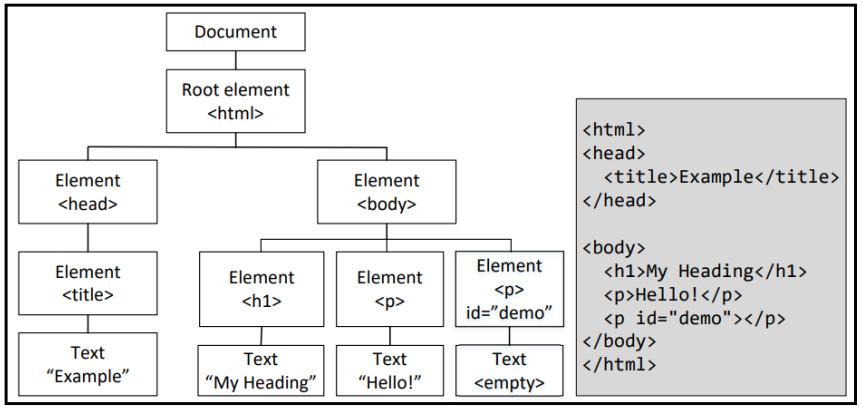


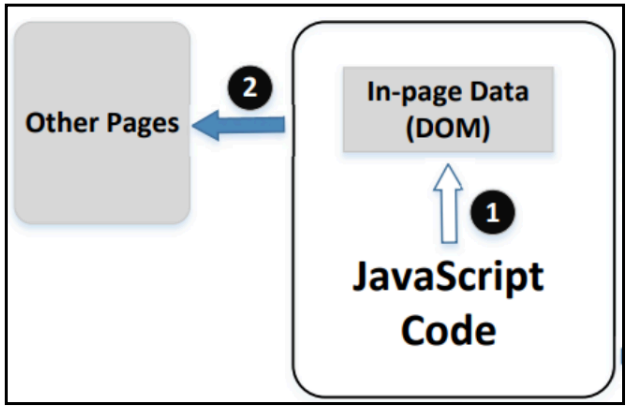

```
<html>
<head>
  <title>Example</title>
</head>

<body>
  <h1>My Heading</h1>
  <p>Hello!</p>
  <p id="demo"></p>
</body>
</html>
```

```
document.getElementById('demo').innerHTML = 'Hello World'
```

2) Restricted Access to DOM of other pages

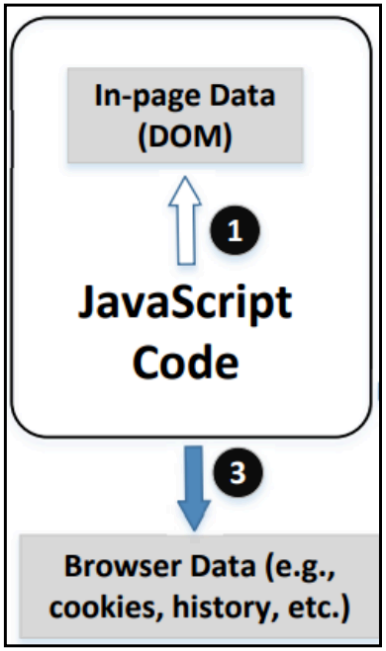
- Javascript gets a handler to root node of current page's DOM
- Each page has its own DOM if browser opens multiple pages.
  - **Security Policy**: Javascript code on one page cannot access DOM of another page
    E.g., using **document.cookie**, can manipulate all cookies **only** for website where current page comes from (not cookies for other websites)





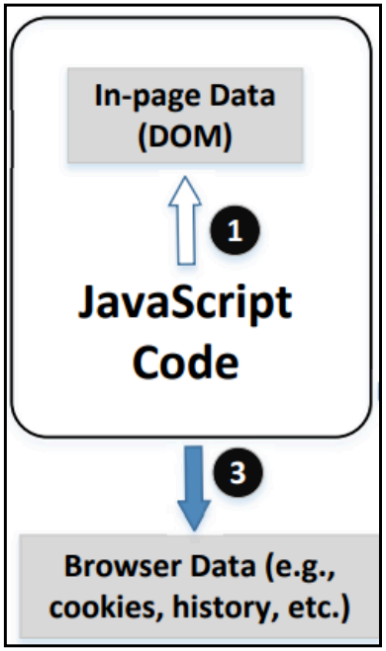
3) Access Browser Data

- Javascript code on page can access browser data via the **window** property but access is controlled
  - Each data type has **different access control models**



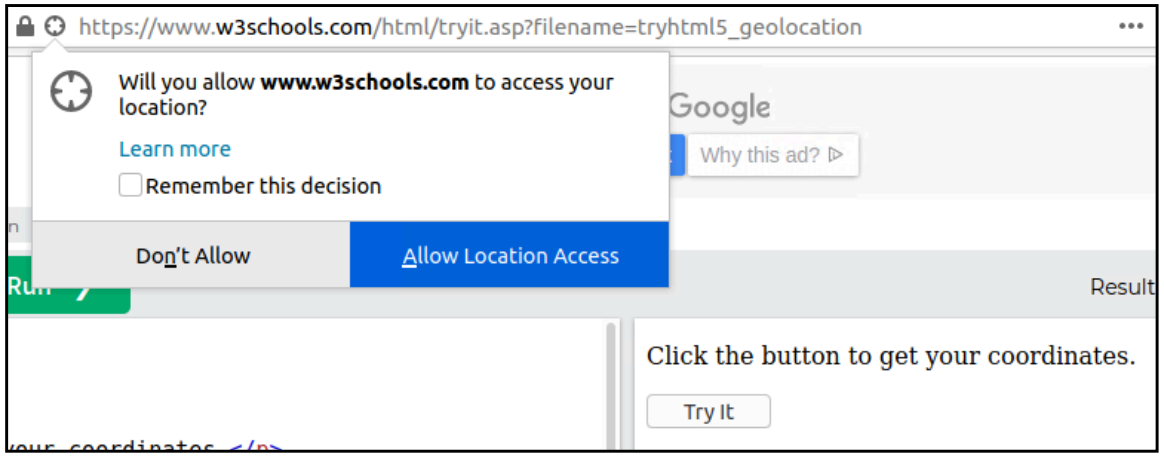
3) Access Browser Data

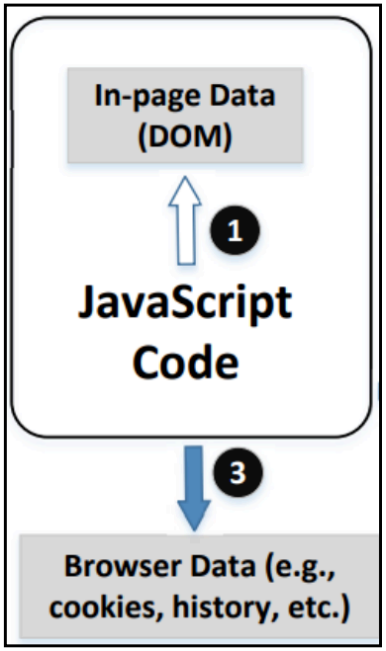
- E.g1, **window.history**
  - Browser searches through browser history and returns **only history of current session**



3) Access Browser Data

- E.g.2, **window.navigator**
  - Returns the browser's location GPS location (if supported)
  - User **must grant permission via a pop-up window**





4) Access File System

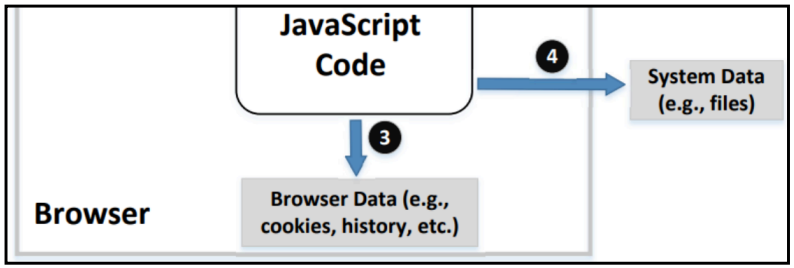
- **Browser Sandbox** prevents JavaScript code from directly accessing local file system
- User needs to grant permission via file selection (or manual drag/drop)

```
<input type="file" id="file-selector">
```

File selection: **grant permissions by selection**

```
var files = document.getElementById('file-selector').files;
```

Get the file handlers (for all files selected) using DOM API



4) Access File System

- Use handler to read file (bank32/localfiles.html)

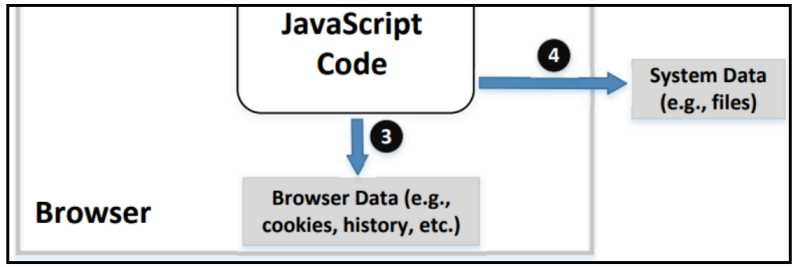
```
var files = document.getElementById('file-selector').files;
if (!files.length) {
   alert('Please select a file!');
   return;
 }
 var reader = new FileReader();
 reader.onloadend = function(evt) {
   if (evt.target.readyState == FileReader.DONE) {
      content = evt.target.result;
      document.getElementById('demo').textContent = content;
   }
 };
 reader.readAsBinaryString(files[0]);
```

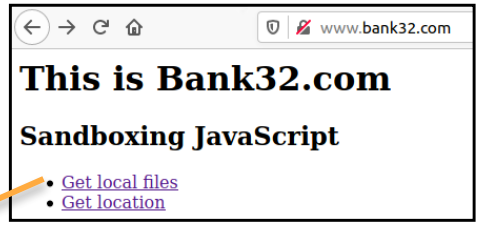
FileReader API

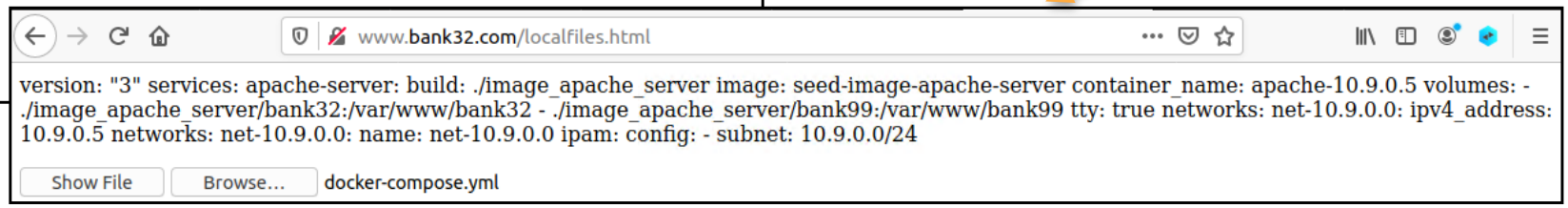
4) Access File System

```
<html>
<body>

<p id="demo"></p>
<button onclick="showfile()">Show File</button>

<input type="file" id="file-selector">

<script>
function showfile() {
   var files = document.getElementById('file-selector').files;
   if (!files.length) {
      alert('Please select a file!');
      return;
    }
    var reader = new FileReader();
    reader.onloadend = function(evt) {
      if (evt.target.readyState == FileReader.DONE) {
         content = evt.target.result;
         document.getElementById('demo').textContent = content;
      }
    };
    reader.readAsBinaryString(files[0]);
}
</script>
</body>
```

- E.g., bank32/localfiles.html

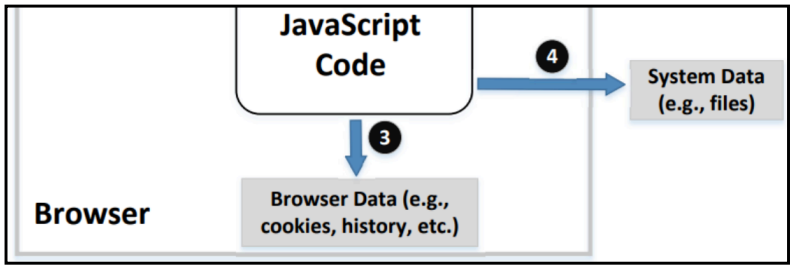






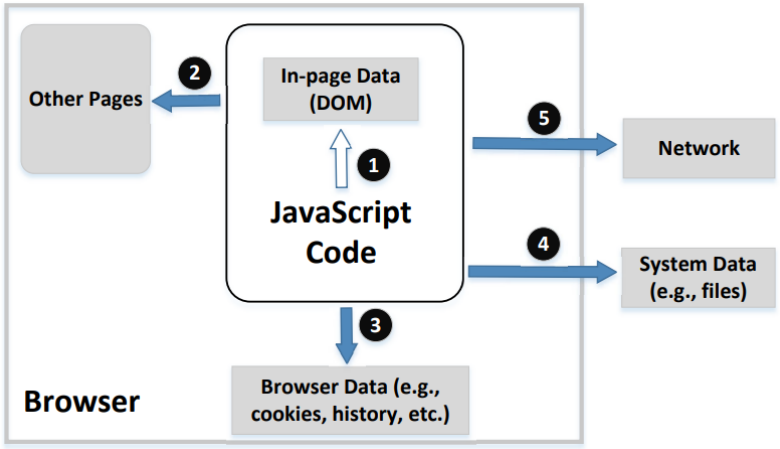
4) Access File System

- **HTML5** supports Filesystem API which allows a web application to interact with local data (e.g., files) in a **sandboxed** manner
  - Each web application has its own filesystem
  - Cannot access random folders on local machine



5) Access Network and Ajax

- **Cannot** directly access network via **Socket APIs**
  - I.e., cannot simply remote connect using an arbitrary port number of standard port that is not allowed (e.g., ftp)
- Three communication mechanisms
  - Normal HTTP request
  - Ajax
  - WebSocket
- Security policies are different



Asynchronous **J**avaScript **a**nd **X**ML (**Ajax**)

- Process:
  - Send a typical HTTP request
  - Data in response is given to a callback function (if any), that will handle the data and update the DOM or content of current page (**no page reload is needed**)
- Slowly being **superseded by the Fetch API**
  - Fetch API provides a more powerful and flexible feature set used for fetching resources (including network resources)

Ajax Example

- Fetch time from server and interactively displays it on page

```
<html>
<body>
<h1>Ajax Experiment</h1>

<div id="demo">Placeholder for data</div>

<script type="text/javascript">
function send_ajax()
{
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "http://www.bank32.com/getdata.php", true);
  xhttp.send();
}
</script>
<p/>
<button type="button" onclick="send_ajax()">Send Ajax</button>
</body>
</html>
```

Callback function

getdata.php

```
<?php
 echo "Time from Bank32: ".date("h:i:sa")
?>
```

Same Origin Policy on Ajax

- Security policy applied by default
- Only responses coming from www.bank32.com are given to the Ajax callback function if page holding the javascript code also comes from www.bank32.com

ajax32.html

```
<html>
<body>
<h1>Ajax Experiment</h1>

<div id="demo">Placeholder for data</div>

<script type="text/javascript">
function send_ajax()
{
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "http://www.bank32.com/getdata.php", true);
  xhttp.send();
}
</script>
<p/>
<button type="button" onclick="send_ajax()">Send Ajax</button>
</body>
</html>
```

Request to same server as current page: no problem

Same Origin Policy on Ajax

- Security policy applied by **default**
- Page from www.bank32.com trying to access www.bank99.com (using Ajax)

code in ajax32-cross-origin.html or **ajax99**.html

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    document.getElementById("demo").innerHTML = this.responseText;
  }
};
xhttp.open("GET", "http://www.bank99.com/getdata.php", true);
xhttp.send();
```

```
<?php
 #header("Access-Control-Allow-Origin: http://www.bank32.com");
 echo "Time from Bank99: ".date("h:i:sa")
?>
```

Same Origin Policy on Ajax

- Page from www.bank32.com trying to access www.bank99.com (using Ajax)
  - A cross origin request…. will fail

What is Blocked: Request or Response?

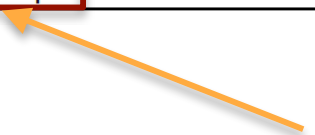
- Request was sent out, response came back
- Browser blocks Ajax code from accessing the response

```
[04/07/23]seed@VM:/$ sudo tcpdump -i br-5929bd713a0c -n -v
    10.9.0.1.54928 > 10.9.0.5.80: Flags [P.], cksum 0x1589 (incorrect -> 0xb983), seq 1:332, ack 1, win
502, options [nop,nop,TS val 1077483948 ecr 3061407169], length 331: HTTP, length: 331
        GET /getdata.php HTTP/1.1
        Host: www.bank99.com
        User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
        Accept: */*
        Accept-Language: en-US,en;q=0.5
        Accept-Encoding: gzip, deflate
        Origin: http://www.bank32.com
        Connection: keep-alive
        Referer: http://www.bank32.com/ajax32-cross-origin.html
```

What is Blocked: Request or Response?

- Request was sent out, response came back
- Browser blocks Ajax code from accessing the response

```
[04/07/23]seed@VM:/$ sudo tcpdump -i br-5929bd713a0c -n -v
    10.9.0.5.80 > 10.9.0.1.54928: Flags [P.], cksum 0x1529 (incorrect -> 0x1add), seq 1:236, ack 332, wi
n 507, options [nop,nop,TS val 3061407170 ecr 1077483948], length 235: HTTP, length: 235
        HTTP/1.1 200 OK
        Date: Fri, 07 Apr 2023 12:20:16 GMT
        Server: Apache/2.4.41 (Ubuntu)
        Content-Length: 31
        Keep-Alive: timeout=5, max=100
        Connection: Keep-Alive
        Content-Type: text/html; charset=UTF-8

        Time from Bank99: 12:20:16pm
```



Response is present but is blocked by browser

Why Blocking the Response?

- **Cross-Origin** access compromise privacy
  - Same-origin policy is enforced
- Example: Ajax code in Facebook page
  - allowed to access the user's Facebook data
  - not allowed to access the user's Google data

Relaxing the Restriction

- The same-origin policy is too restrictive
- **CORS** (Cross-Origin Resource Sharing)
  - **Whitelist** provided **by server**: grant permissions
- **CORS** policy on www.bank99.com

bank99/getdata.php

```
<?php
 header("Access-Control-Allow-Origin: http://www.bank32.com");
 echo "Time from Bank99: ".date("h:i:sa")
?>
```

Relaxing the Restriction

- Response from www.bank99.com now contains new header

```
[04/07/23]seed@VM:/$ sudo tcpdump -i br-5929bd713a0c -n -v
    10.9.0.5.80 > 10.9.0.1.55698: Flags [P.], cksum 0x155d (incorrect -> 0x1eaf), seq 1:288, ack 332, wi
n 507, options [nop,nop,TS val 3078932559 ecr 1095009337], length 287: HTTP, length: 287
        HTTP/1.1 200 OK
        Date: Fri, 07 Apr 2023 17:12:21 GMT
        Server: Apache/2.4.41 (Ubuntu)
        Access-Control-Allow-Origin: http://www.bank32.com
        Content-Length: 31
        Keep-Alive: timeout=5, max=100
        Connection: Keep-Alive
        Content-Type: text/html; charset=UTF-8

        Time from Bank99: 05:12:21pm
```

WebSocket

- Ajax uses HTTP: half-duplex
  - Browser sends request, server responds
  - No “push” mechanism (from server)
- WebSocket is **full-duplex**
  - Both browser/server can send data (without request)
- wss: ws built on top of TLS

Security Policy on WebSocket

- Browser does not restrict data from WebSocket
  - Different from Ajax (where access control is implemented on client side)
  - i.e., No **client-side** same origin policy (like in Ajax) exits
- Access control is conducted on server side
  - Server must manually check "Origin" header in the request (i.e., **server-side** policy)

```
    10.9.0.1.54928 > 10.9.0.5.80: Flags [P.], cksum 0x1589 (incorrect -> 0xb983), seq 1:332, ack 1, win
502, options [nop,nop,TS val 1077483948 ecr 3061407169], length 331: HTTP, length: 331
        GET /getdata.php HTTP/1.1
        Host: www.bank99.com
        User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
        Accept: */*
        Accept-Language: en-US,en;q=0.5
        Accept-Encoding: gzip, deflate
        Origin: http://www.bank32.com
        Connection: keep-alive
        Referer: http://www.bank32.com/ajax32-cross-origin.html
```

Cable Haunt Attack

- Discovered: January 2020
- Affected many Broadcom-based cable modems
- These modems run a **WebSocket**-based server program
  - Server program did not check for origin header to prevent unauthorized origins
  - JavaScript code from malicious website can interact with the server: a door is open
  - Attacker exploited a buffer overflow vulnerability on the server to inject shellcode into the cable modem (full remote access)