Web Security

Güneş Acar & Erik Poll

Digital Security group
Radboud University Nijmegen

This course

The web is a endless source of security problems. *Why?*

- The web is very widely used, so it's *interesting* to attack
- The web is very COMPLEX and continuously evolving

  so there are *many & often new possibilities* for attacks

Goals of this course:

- *How* do attacks on the web work?
- *What* do they try to achieve?
- *What* can we can do about them?
- *Why* are these attacks possible?

Focus on fundamental concepts, not the latest fashions in web technologies & attacks.

Organisation

Weekly lecture

- Make sure you understand material presented
- Read any reading material mentioned
- Try out the demo webpages mentioned in the lecture
- Ask questions if things are not clear!

Weekly lab session with 3 types of exercises

1. OWASP WebGoat lessons - no need to hand these in
2. challenges at http://websecurity.cs.ru.nl
   handed in automatically; have be done individually
3. ad-hoc assignments - to be handed in (as pair) via Brightspace

Help with lab sessions on Wednesday in terminals rooms

- Work in pairs - discussing with team partner helps!
- Doing the exercises is obligatory to take part in the exam

Cheating is trivial, but exam will assume familiarity with the exercises

Course materials

All info & course material is in Brightspace

Obligatory reading

- All the slides
- ‘Understanding The Web Security Model’ by Eric Rescorla
- Some articles & blog posts linked to in Brightspace

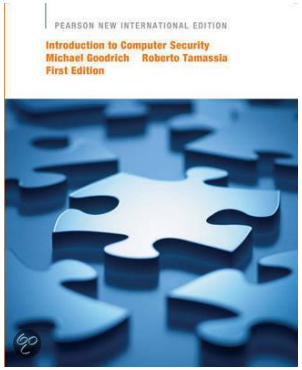
Optional background reading:

Introduction to Computer Security

by Michael Goodrich and Roberto Tamassia

Chapters 1, 5.1, 7

There is a copy in the studielandschap in the library



Any questions on organisational matters?

Audience poll (1)

Have you ever built a web site,

or an app that uses web technologies?

(eg. HTTP, HTML, JavaScript, XML, JSON)

Audience poll (2)

Have you ever tried to hack a web site?

Audience poll (3)

Have you ever participated in a CTF?

If you like the practical side of this course,

join our student CTF-RU team at https://discord.gg/ducnzsW

Wider context

Security problems arise from attacks on

1. SOFTWARE

2. PEOPLE

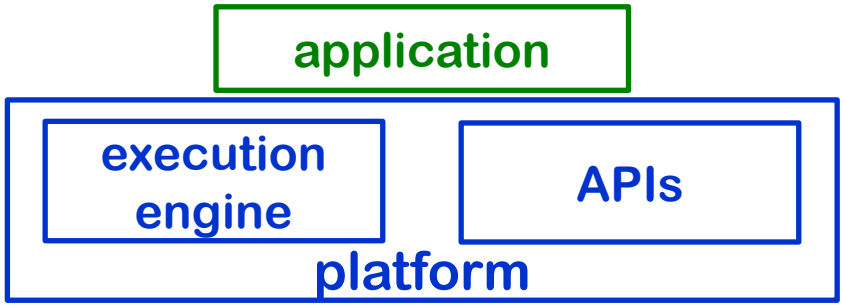
3. the interaction between people & software

Blaming 'stupid users' is usually victim blaming:

if users cannot use a system securely, this is an IT design flaw.

Software & Platforms

Software involves an application running on a platform



Example platforms?

CPU, Windows/Linux/MacOS machine, Android/iOS mobile device, web browser, web server, Amazon/Google/Microsoft Cloud, …

- **CPU & OS is a platform for executing machine code:**
  Topic of *Hacking in C*

- **Browser is platform for rendering HTML and executing JavaScript:**
  Topic of this course
  Extra complication: content rendered/executed client-side in the browser is produced by & interacts with a server-side application

Today: What is the web?

- Evolution of the web
- Core technologies
  - HTTP
  - URL
  - HTML

    which includes JavaScript & the DOM
- Encodings for representing data
  - base64 encoding, URL encoding, HTML encoding

The internet & the web

The internet & The web

Often confused, but they are different

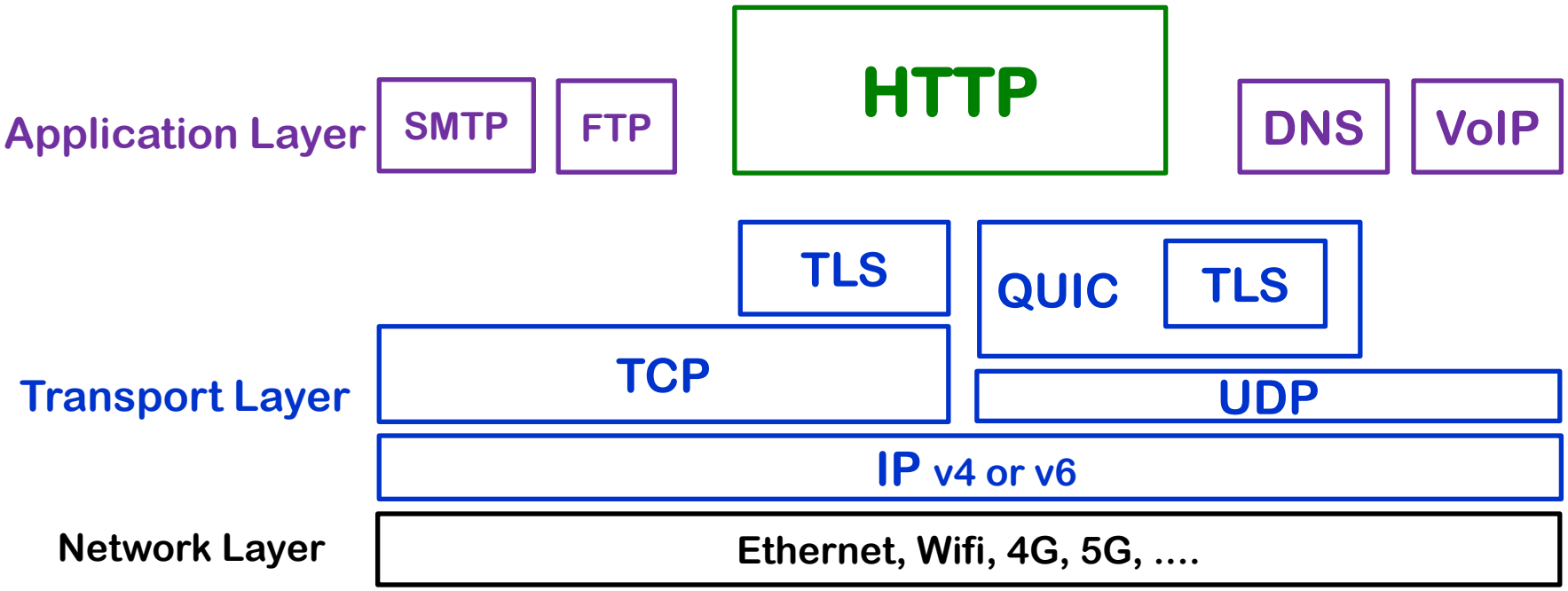
web

internet

- The internet
  - provides network communication between computers
  - using the IP protocol family with UDP or TCP, or QUIC as modern replacement of TCP & TLS
  - using IP addresses to identify computers

- The web
  - set of services that can run *over* the internet
  - using the HTTP/HTML protocol family
  - using domain names and hence DNS to identify services

Protocol stack for internet & web



Various protocols can run over the internet (TCP or UDP) :
email (SMTP), VoIP, ftp, telnet, ssh, ... and HTTP

HTTP uses additional data formats: HTML and URLs

Aside: Protocols

For example: IP, HTTP, HTTPS, DNS, TLS, SMTP, …

Procotol is set of rules for two (or more) parties to interact

- **Not just between computers. People also follow protocols: when they meet, when they buy a coffee, …**

Protocols usually specify two aspects of interaction:

1. **data format for messages**
   - **often specified by regular expression or context-free grammar**
2. **allowed/expected *sequences* of messages**
   - **often specified by finite automaton aka state machine or a Message Sequence Chart (MSC)**

Aside: Languages (or formats)

For example

- file formats: .html, .xml, .js, .json, .docx, .pdf, .txt, .mp3, .jpeg, .mp4, …
- other pieces of data: URLs, domain names, email addresses, IP packets, HTTP responses & requests, …

The definition of a language or data format involves

- syntax
  - what are correct words/sentences/sequences of bytes?
- semantics
  - what do these mean?

    ie. how should they be interpreted/processed?

Complexity and ambiguity in (large number of!) languages are major root causes of security problems

The world wide web

The web is one of the services available over the internet

www = HTTP + HTML + URLs

HT in HTTP and HTML stands for HyperText

At the server side, it involves a web server that typically

- listens to port 80
- accepts HTTP requests (eg `GET` or `POST` request), processes these,
  and then returns HTTP responses

At the client side, it involves web browser or app

Evolution of the web

Evolution of the web

Web is constantly evolving

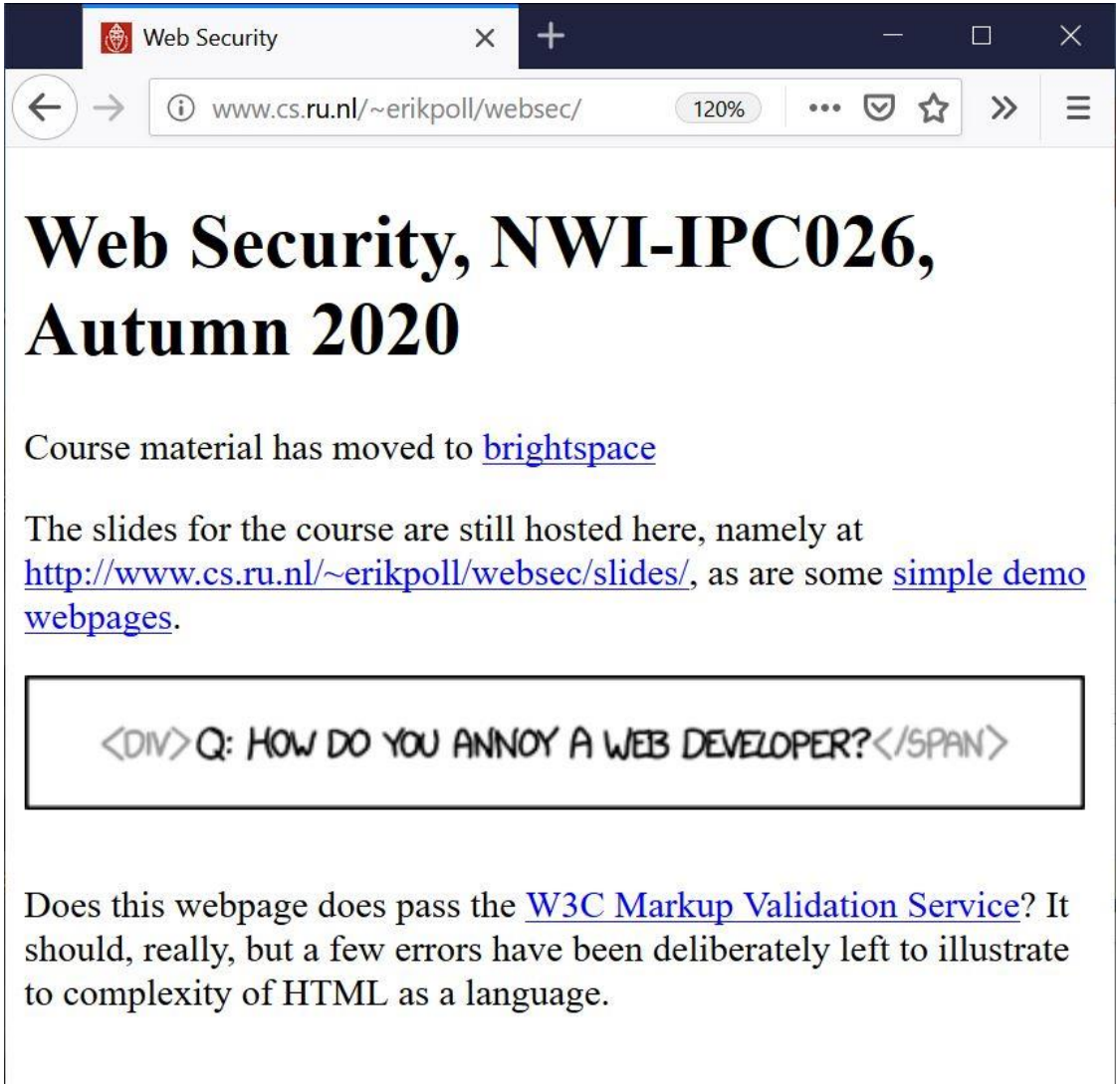
- more functionality, more flexibility, nicer GUIs, … ☺
- more complexity, more & new security problems, … ☹

Stages in the evolution:

1. Static hypertext files
2. Dynamically generated web pages (incl. Web 2.0)
3. Dynamic web pages aka web apps
4. Servers provides APIs for more fine-grained interaction than whole webpages
5. More Web APIs in browsers
6. Apps on mobile phones & tablets
7. Web-based applications on laptops & desktops

1. Static hypertext

For example, http://www.cs.ru.nl/~erikpoll/websec/index.html



1. Static hypertext files

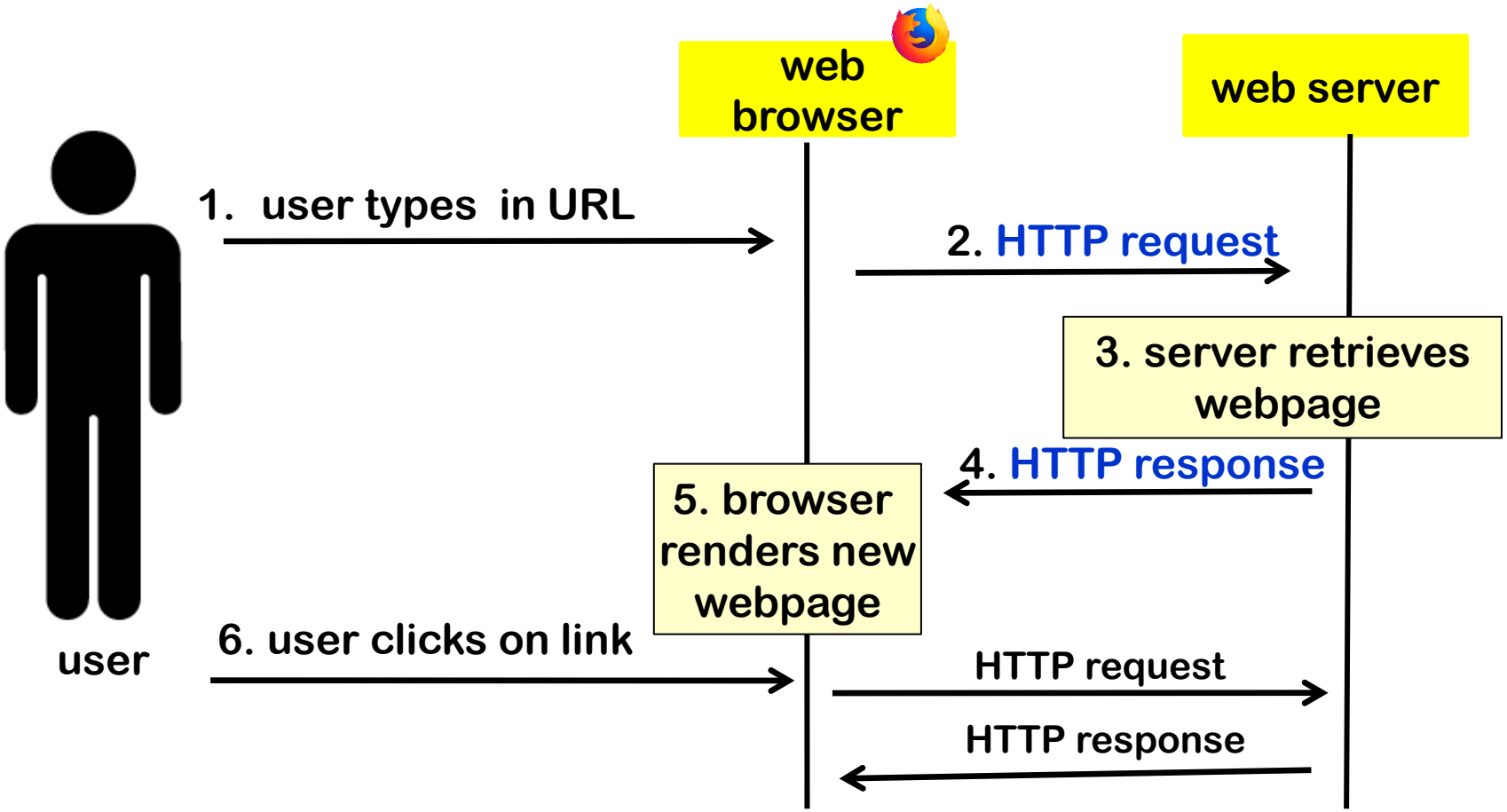
Originally, the web consisted of static HTML:
hypertext with links and pictures

Eg `http://www.cs.ru.nl/~erikpoll/websec/index.html`

- The webpage can simply be a fixed .html file on the file system; a (very simple) web server only has to retrieve files from disk
- The webpage does not depend on user input & is not personalised: all users see the same page.
- No user interaction, apart from the user clicking on links to load another page

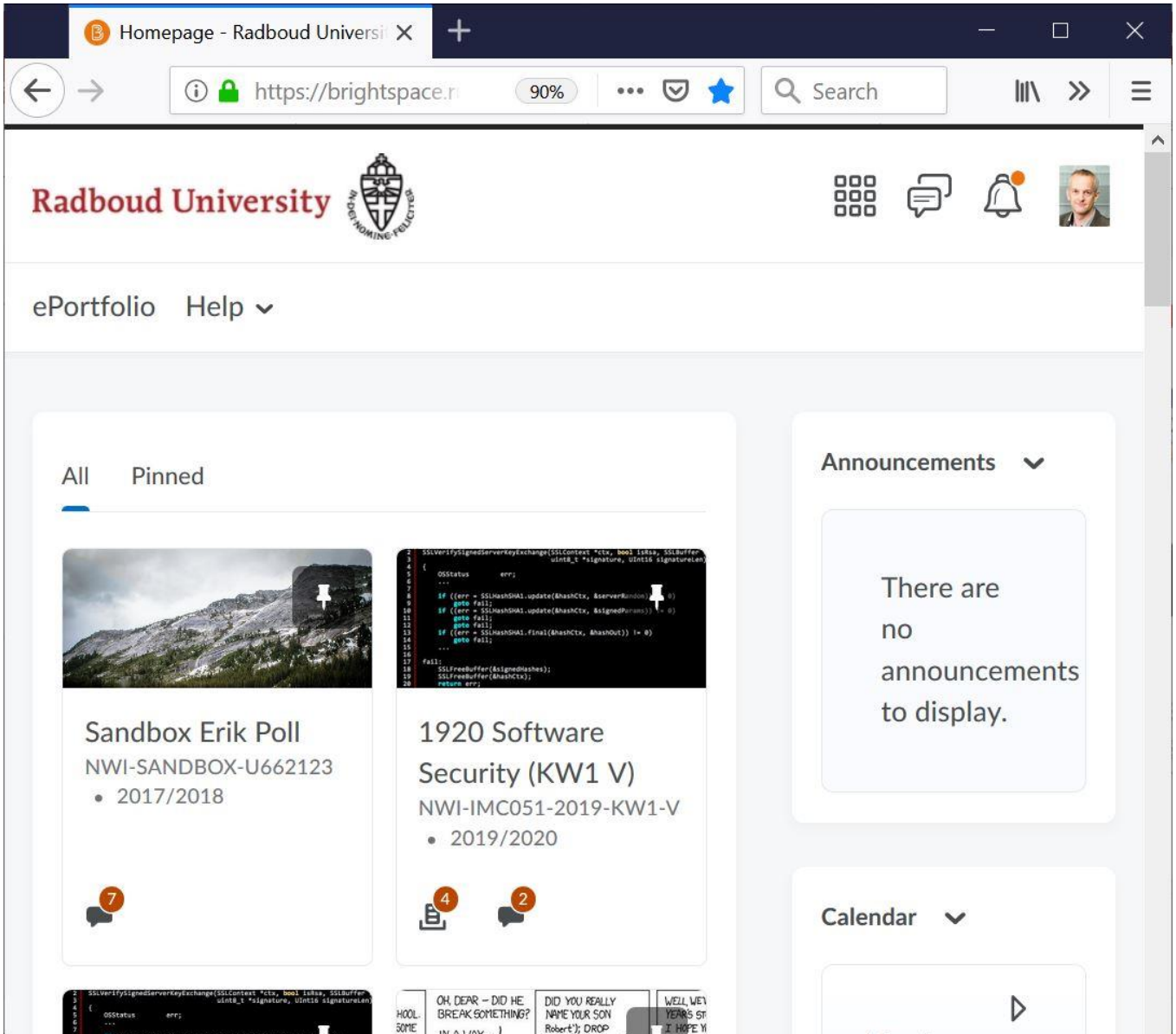
- Maintaining large sets of .html files quickly becomes a pain, and then using Content Management System (CMS) is better

Synchronous interaction on the web

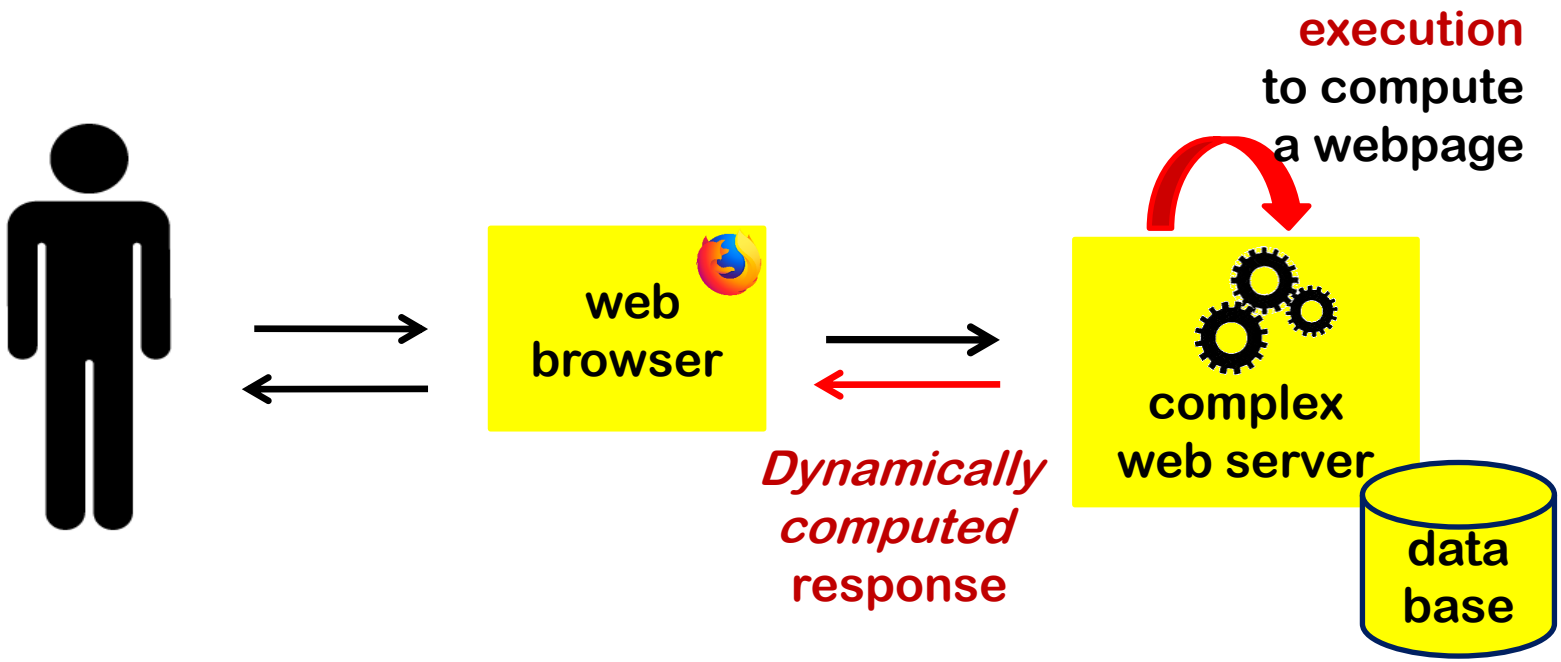


This is oversimplified. Even simple browsing is much more asynchronous. E.g. browser will start rendering while images are retrieved.

2. Dynamically created web pages
– ie. server side execution



2. *Dynamically created* web pages



Interaction still synchronous

In general, having execution is nice, as it is flexible & powerful but this also makes it DANGEROUS

2. *Dynamically created* web pages

Different users will be served a different webpage.

Eg Google, Gmail, Facebook, Brightspace, persoonlijkrooster.ru.nl, …

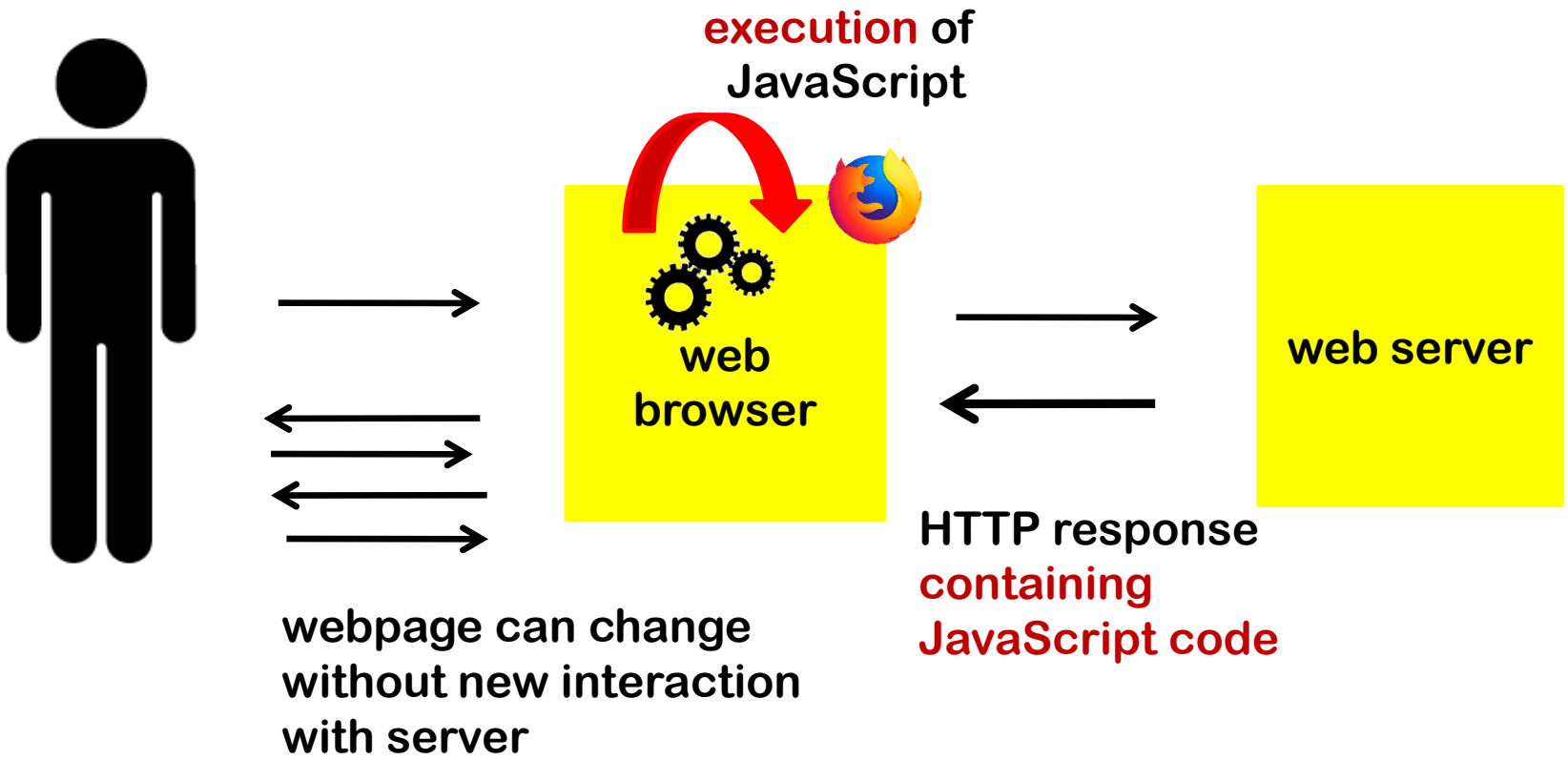
This enabled web 2.0 with user-generated content:

web forums, Wikipedia, social media: Facebook, Instagram, Twitter, …

More terminology:

- Web server that simply *retrieves* HTML files now becomes web application that *generates* HTML content
- This application can run on web application server (eg Apache Tomcat, IBM WebSphere,…) or be invoked using CGI
- HTML generation can be done with templates or fully programmatic
  - using web templating language and associated engine
  - using (general purpose or dedicated) scripting or programming languages eg Perl, Python, PHP, Java, C#, Ruby on Rails, Go, JavaScript …

3. Dynamic web pages aka web *apps*
= ie client-side execution



Eg. http://www.cs.ru.nl/~erikpoll/websec/demo/demo_javascript.html

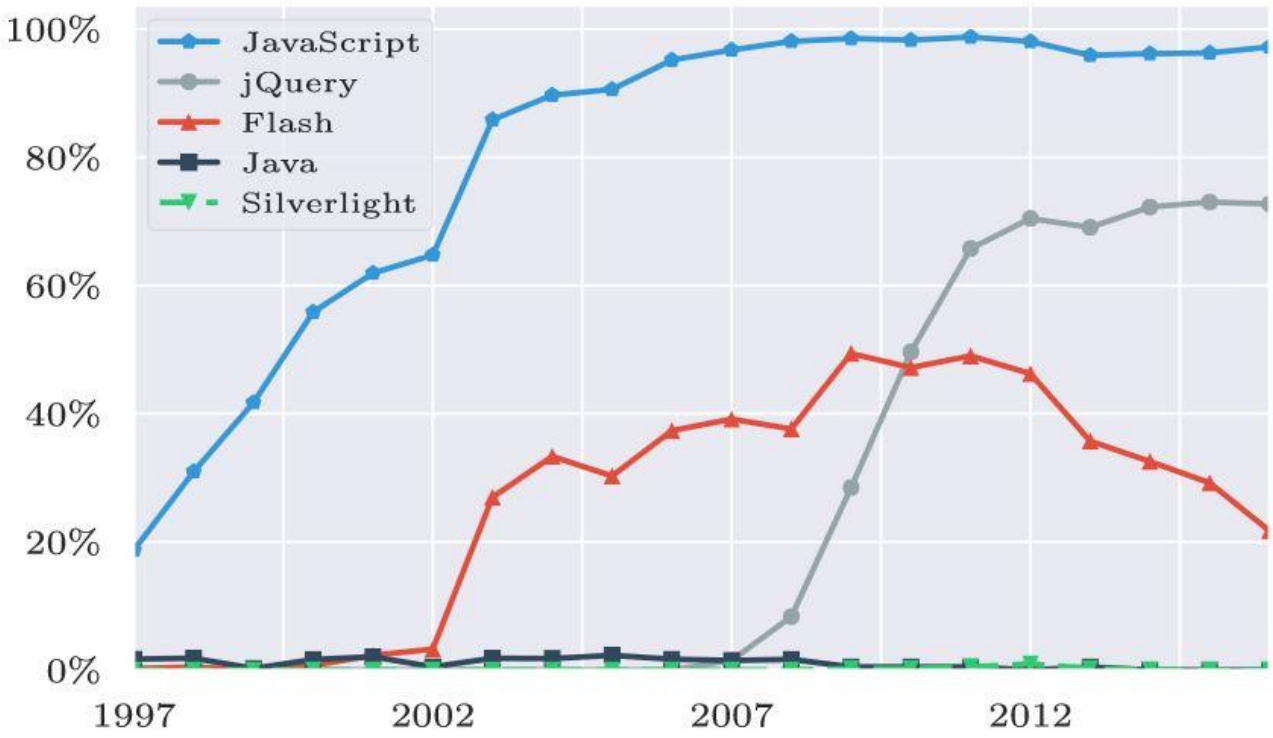
3. *Dynamic* web pages – ie client-side execution

- Web page includes code that is executed *in the browser*
- Browsers offers APIs that such code can use, incl. the DOM API
- Two main programming languages for this:
  - JavaScript
    - part of the HTML standard since HTML5
  - WebAssembly (Wasm)
    - since 2017

- Goals:
  - more attractive web pages
  - more and faster interaction with the users

- Older languages used for dynamic behavior in the browser included Java, ActiveX, Flash, Silverlight, …

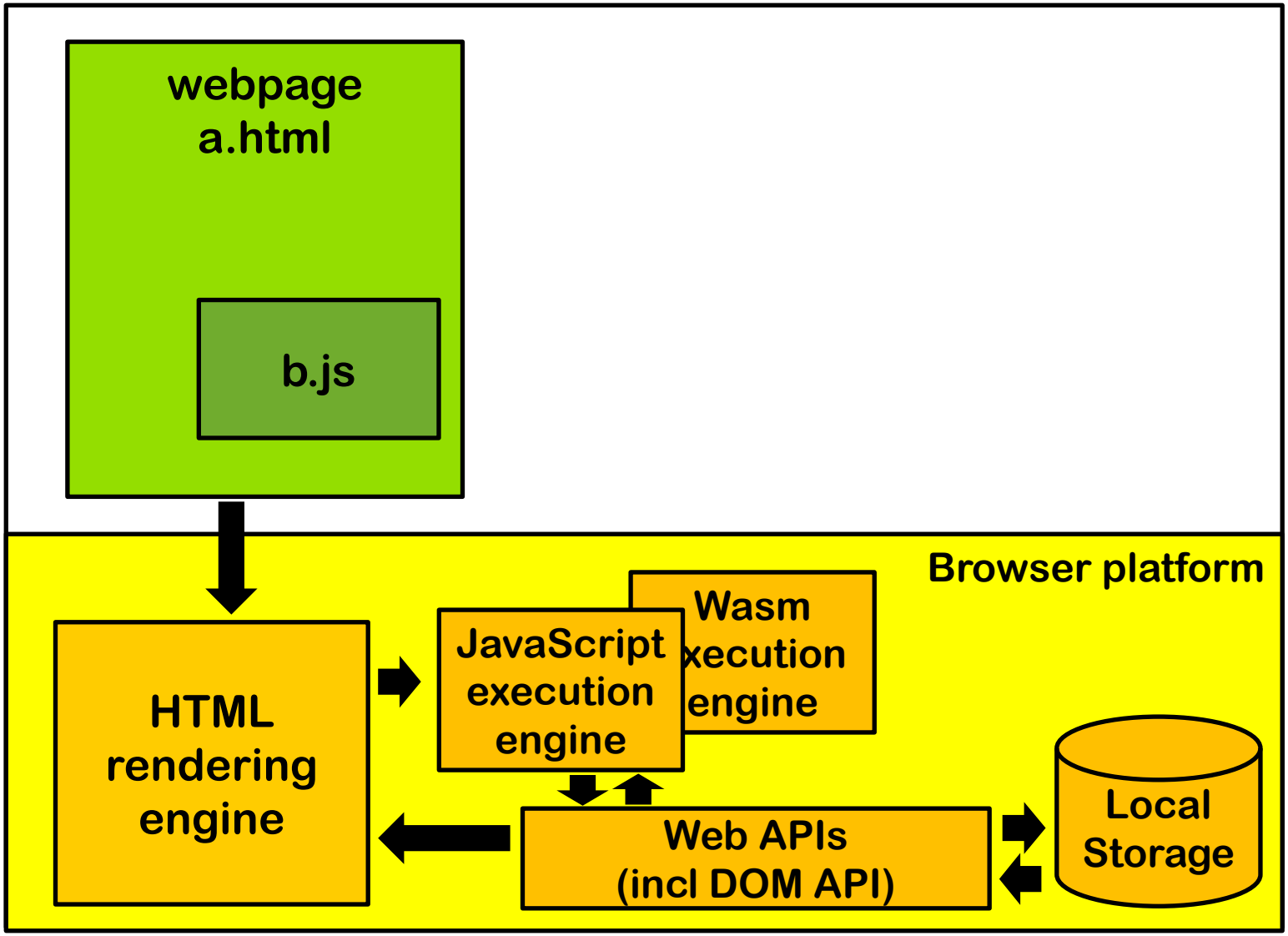
Evolution in web technologies



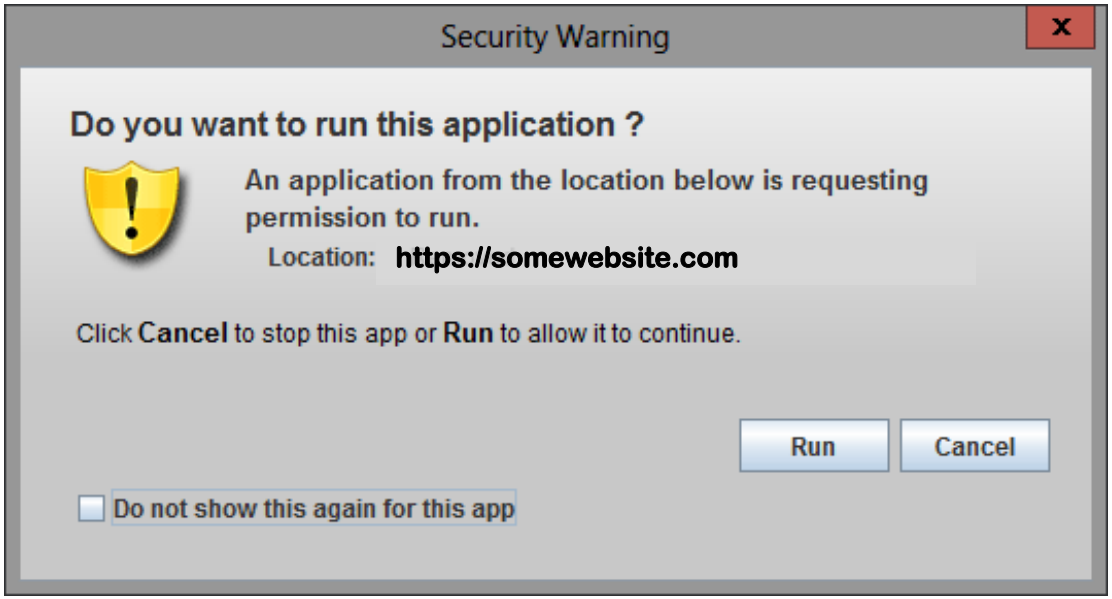
Technologies used by top 500 web sites

[Source: Stock et al, How the Web Tangled Itself: Uncovering the History of Client-Side Web (In)Security, USENIX Security Symposium, 2017]

Inside the browser



Isn't downloading & running code a security risk?



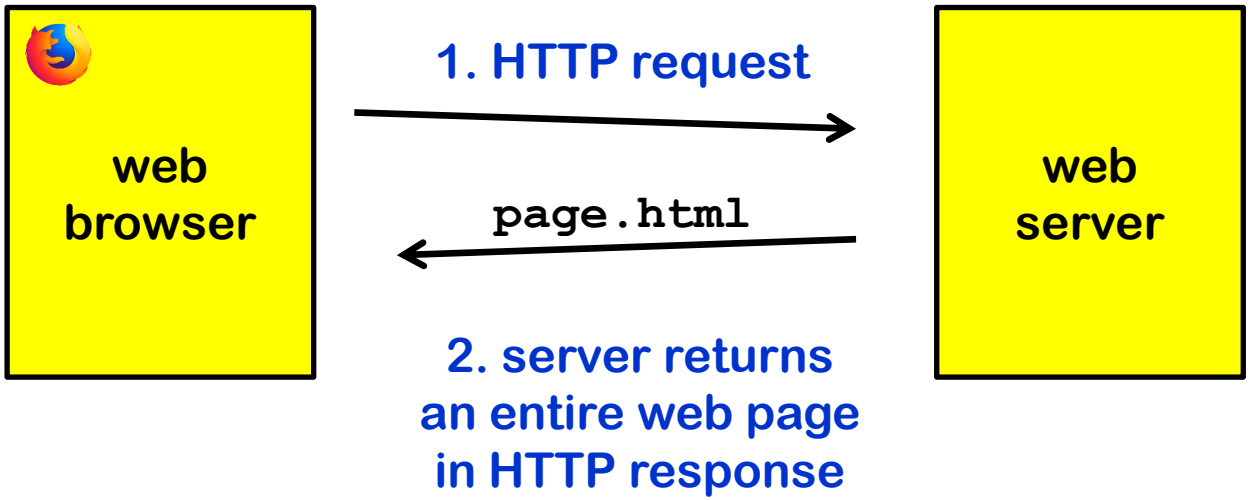
Running random applications on your computer is a bad idea!

Running downloaded code inside your browser happens all the time!

(Why) is this ok?

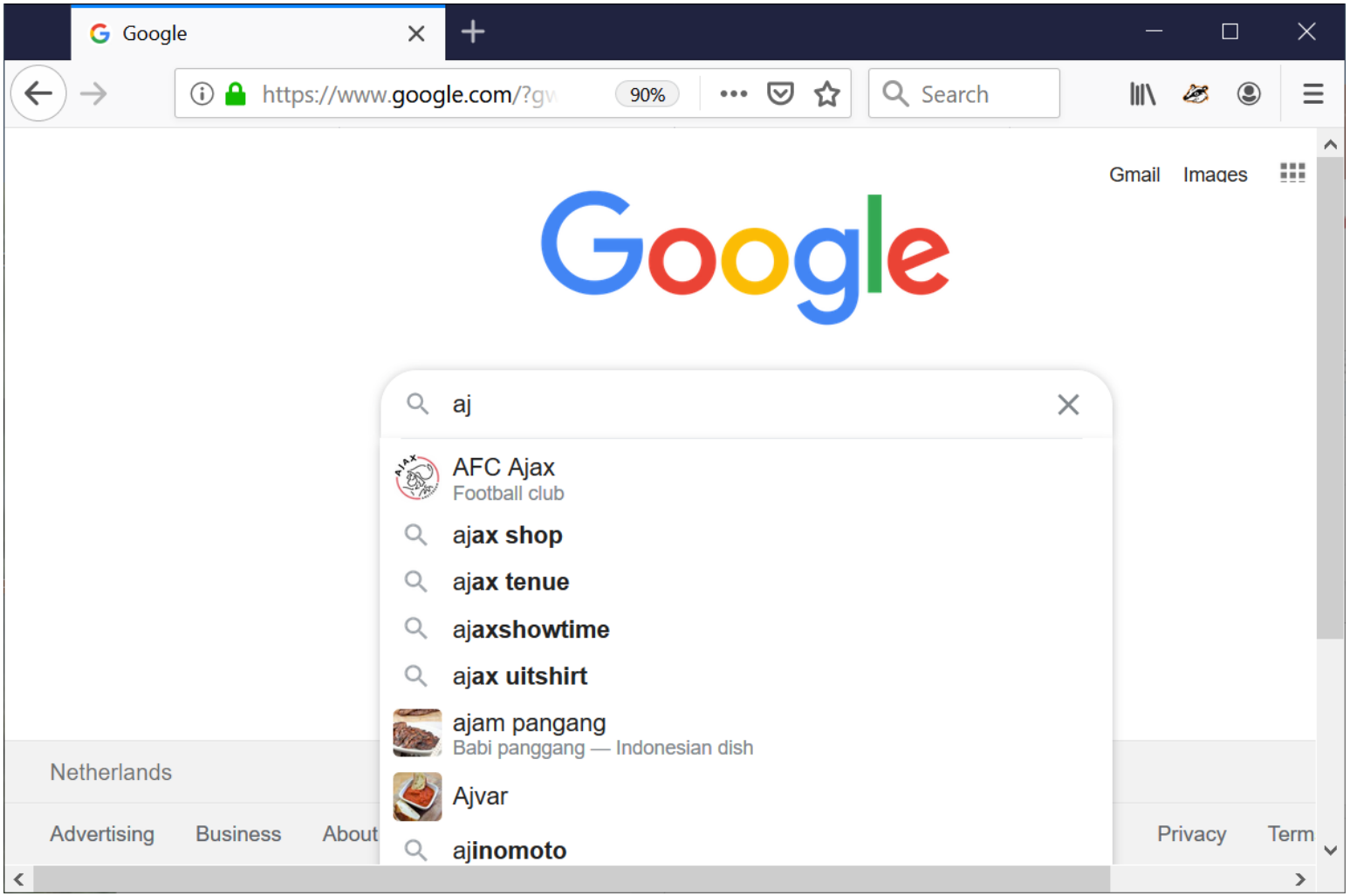
JavaScript code executes in a tightly controlled sandbox inside the browser

So far: web server provides entire web page



Next big step in evolution of the web: web servers offering APIs (aka Web Service APIs) for smaller snippets of information.

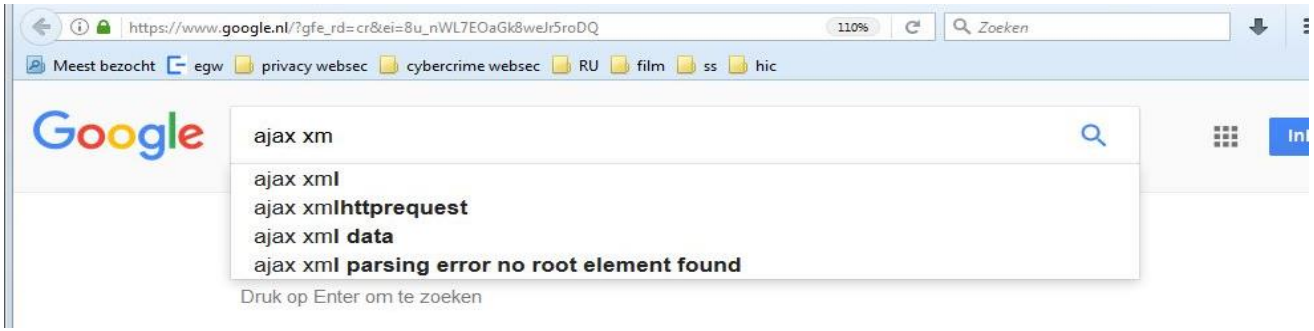
4. Asynchronous interaction with Ajax



4. Ajax = Asyncronous JavaScript with XML

JavaScript in browser asynchronously interacts with the server, using a `XMLHttpRequest` object

Classic example: word completion in Google search bar as you type

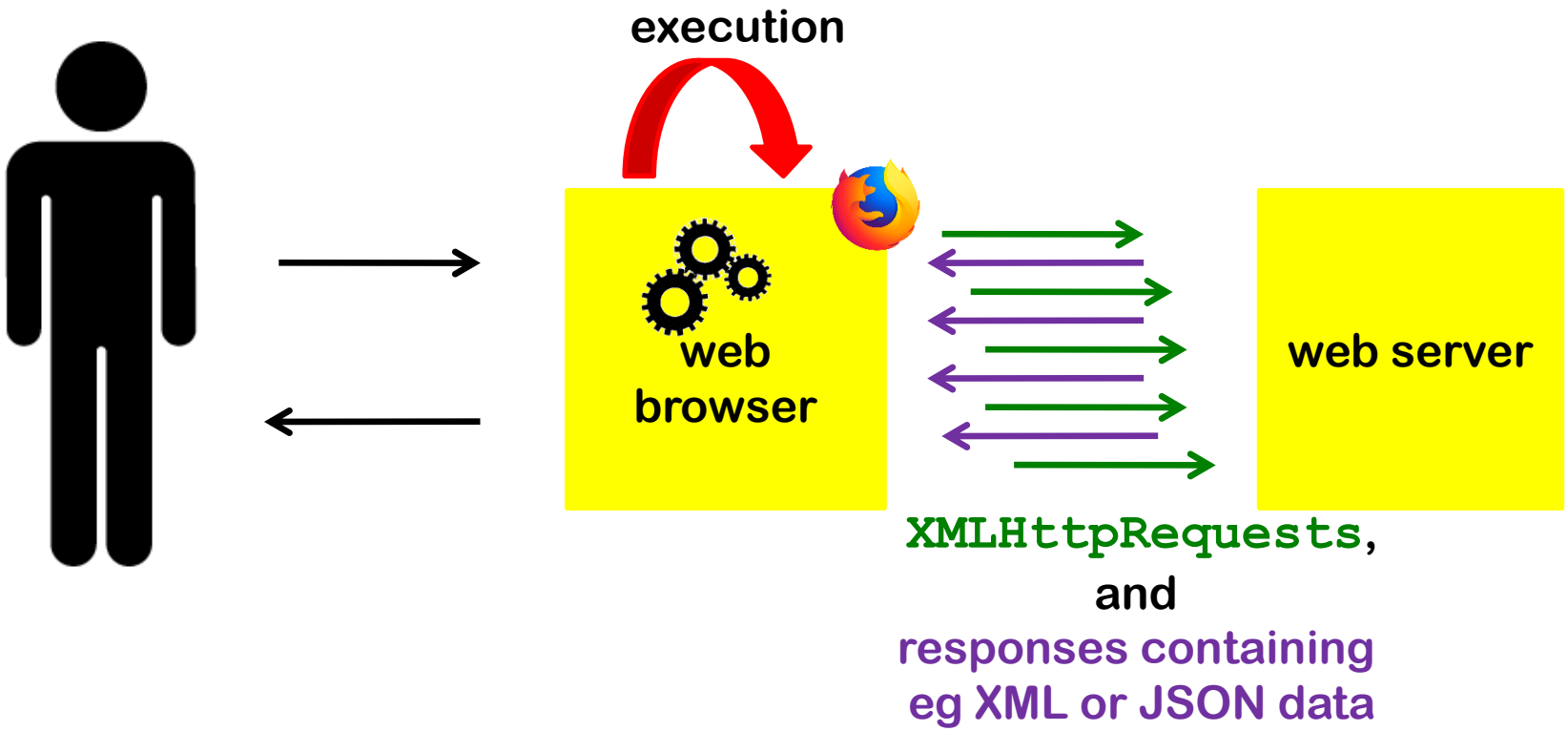


Typical characteristics

1. interaction independent of the user clinking on links
2. without reloading *whole* webpage: client-side JavaScript code updates *part* of webpage

Originally, the data exchanged was in XML format, nowadays JSON is more commonly used.

4. Asynchronous interaction with Ajax



With Ajax the initiative for interaction still lies with the browser

With `WebSockets` communication becomes full duplex

ie. web server can take initiative to send messages

XML & JSON

Extensible formats for exchanging data between browser and server

- **XML (eXtensible Markup Language)**

```
<students> <student>  <firstName>John</firstName>
                       <lastName>Doe</lastName> </student>
           <student>  <firstName>Jan</firstName>
                       <lastName>Jansen</lastName></student>
</students>
```

- **JSON (JavaScript Object Notation)**

```
{"students":[
    { "firstName":"John", "lastName":"Doe" },
    { "firstName":"Jan", "lastName":"Jansen" }
  ] }
```

Lots of debate about pros and cons of XML vs JSON.

JSON less verbose & closer to JavaScript and favourite these days.

HTML vs XML (& JSON)

- HTML tags are fixed and e.g. define how information should be grouped or displayed

```
<h1>This is a heading</h1>
<p>This is a paragraph of text. It ends here.</p>
```

- XML can be extended with tags customised to include semantic information, eg.

```
<date>1/9/2020</date>
<price>3.20 euro</price>
<studentnumber>s123456</studentnumber>
```

=

Web 3.0?

- In the early 2000s, some people did a proposal for Web 3.0, aka the Semantic Web, where more data would have meaningful tags like on the previous slide, to facilitate automated processing
  - eg web scraping would become a lot easier

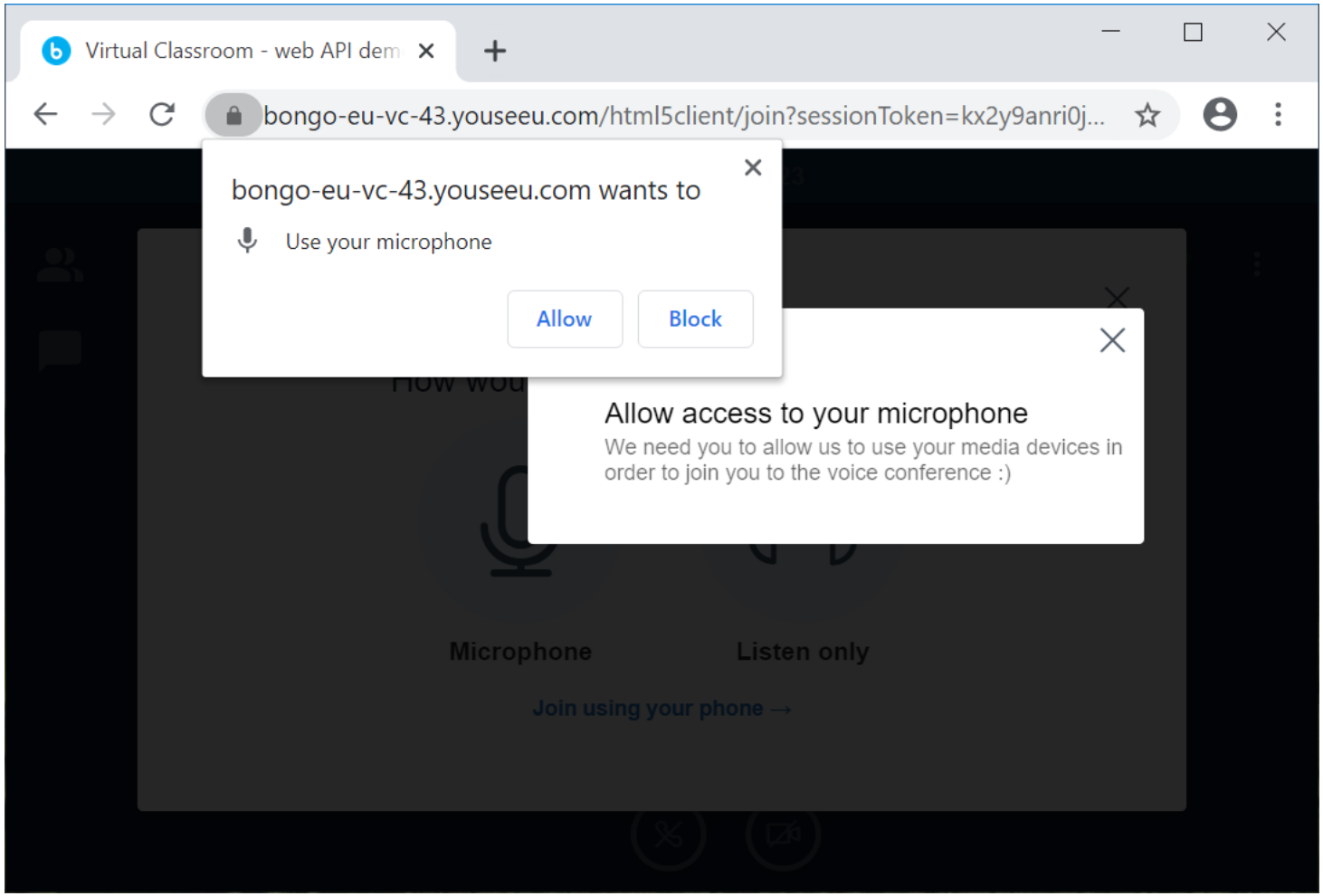
It never took of...

- In the late 2010s, some people proposed a Web 3 which would be some vague combination of the web with blockchain, NFTs, and maybe a metaverse.

This is just some vague hype or scam and a big disaster

See https://web3isgoinggreat.com/

5. More Web APIs in browsers



5. More Web APIs in browsers

Via Web APIs the browser provides functionality to JavaScript and Web Assembly in web pages.

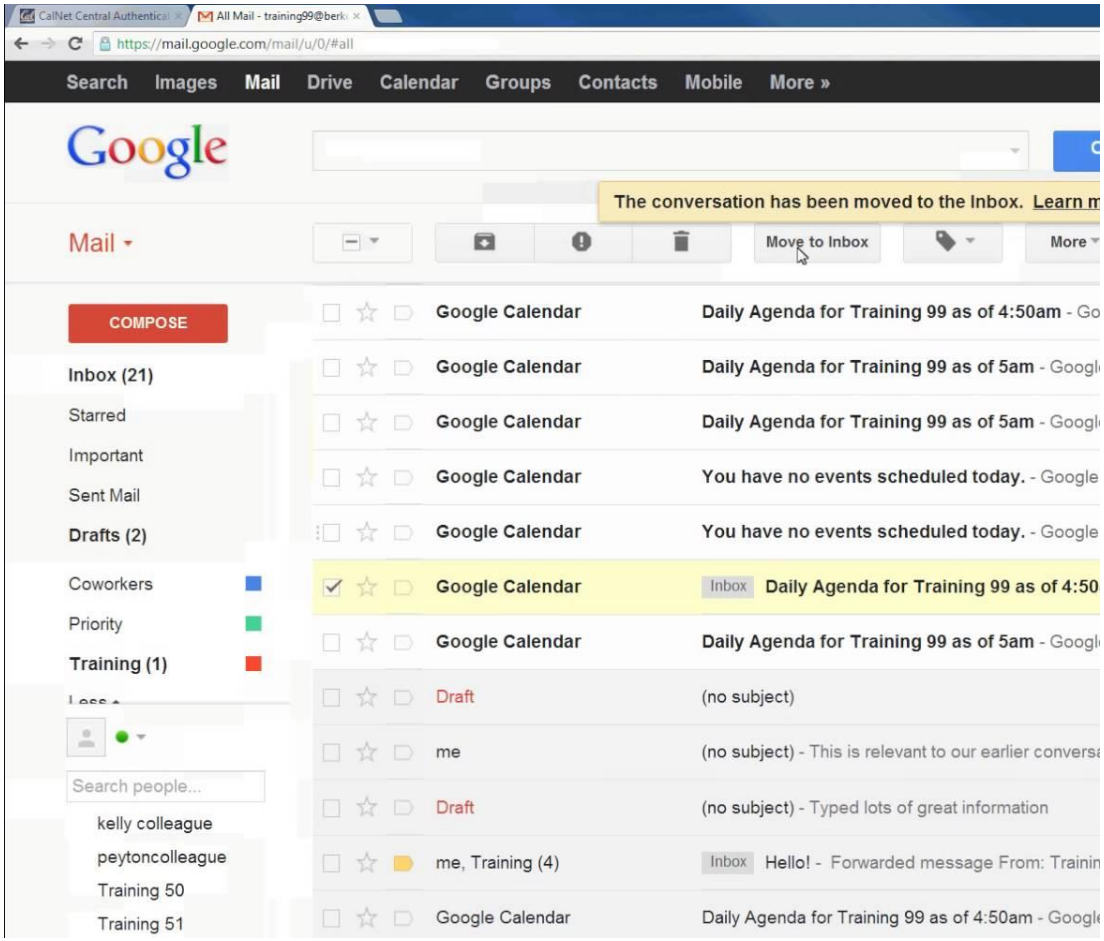
The set of Web APIs is constantly evolving, with some differences between browsers.

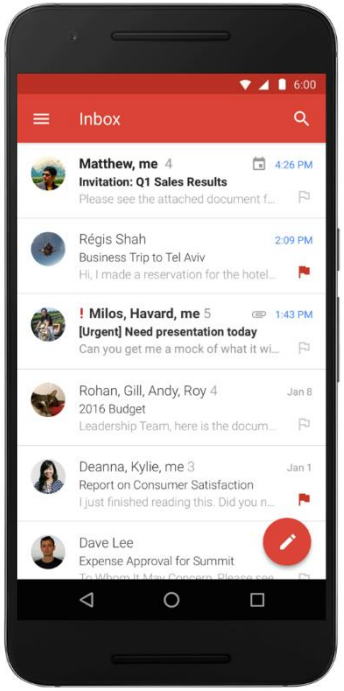
- **Many Web APIs have been added over the years:**
  for sound, accessing web cam, microphone, allowing screen sharing, using local storage on the computer, …

- **The first Web API, the DOM API, allows interaction with the webpage itself**
  - **Eg http://www.cs.ru.nl/~erikpoll/websec/demo/demo_DOM.html**
  - **Lot of examples in later lectures**

See https://developer.mozilla.org/en-US/docs/Web/API for full list of Web APIs

6. From web app in browser to app





6. Apps on mobile phones & tablets

Instead of one generic *browser* to access *many* services,
a dedicated *app* for *one* service

App can still use HTTP, HTML, XML, JSON,…
App and browser can talk to the same server
Under the hood, many apps use the same HTML rendering engine, eg WebKit, as used in browsers.

Some apps are simply stand-alone dedicated browsers that display HTML contents.

- Advantages
  - Easy to port from iOS to Android and vv.
  - Content of the webpage can be reused for the app
  - Programmers familiar with web technologies can easily built apps

7. Deskop apps using web technologies

Many desktop applications on your laptop also make use of HTTP and HTML to interact with a server

- possibly the same server that web apps and mobile apps also interact with

  Eg you can run Discord 1) in your browser, 2) as standalone app on your phone, or 3) as standalone application on your laptop

Eg Electron apps built using HTML and JavaScript.
See https://www.electronjs.org

Same tech(nology) stack
=> same security problems

Tech stack includes HTML renderer & JS engine

Core web technologies:

Protocols,
Languages,
Encodings

Background: IP

IP (Internet Protocol) is the protocol to route data from source node to destination node

- on best effort basis: no guarantee that data will arrive

Most important transport layer protocols on top of IP

- TCP
  - establishes connection, ie sequence of data packets
  - requires set-up, but then guaranteed delivery, in the right order
- UDP
  - connection-less, separate data packets
  - no set-up, by no delivery guarantees

Nodes are identified by IP addresses

- 32 bit for IPv4, 128 bit for IPv6

DNS protocol translates logical domain names to IP addresses

Background: RFCs

Internet-related protocols and formats defined in RFCs (Requests For Comments).

RFCs become standards when approved by the Internet Engineering Task Force.

Eg, the official standard for IP is defined in RFC 791 [http://www.ietf.org/rfc/rfc0791.txt]

There are many RFCs, and they can be quite complex!

The World Wide Web Consortium (W3C) defines web-related standards.

URLs

scheme://login:password@address:port/path/to/resource?query_string#fragment
1 2 3 4 5 6 7

1. scheme/protocol name, eg http, https, ftp, file, …
2. credentials: username and password (optional & depreciated!)
3. address: domain name or IP address
4. port: port number on the server (optional)
5. path to the resource (optional)
6. query string: lists parameters param=value (optional)
7. fragment identifier: offset inside web page (optional)

   Fragment id not sent to web server, but processed locally by browser.

The 'living' standard for URLs is at https://url.spec.whatwg.org
Most systems now use this standard instead of the older definition RFC 3986

HTTP

HTTP (Hypertext Transfer Protocol)
used for communication between web browser and web server with HTTP requests and responses.

HTTP requests and responses always consists of three parts:

1. **request or response line**
2. **header section**
3. **entity body**

The browser turns

- **URLs users types**
- **links they click**
- **certain actions of JavaScript in the webpage**

into HTTP requests

HTTP requests

A request has the form

```
METHOD /path/to/resource?query_string HTTP/1.1
HEADER*
BODY
```

HTTP supports many methods. The most important

- GET for information retrieval
  - body usually empty, as any parameters are encoded in URL
- POST for submitting information
  - body contains the submitted information

- XMLhttpRequest for AJAX

HTTP responses

A response has the form

```
HTTP/1.1 STATUS_CODE STATUS_MESSAGE
HEADER*
BODY
```

Important status codes

- 2XX: Success, eg 200 OK
- 3XX: Redirection, eg 301 Moved Permanently
- 4XX: Client side error, eg 404 Not Found
- 5XX: Server side error, eg 500 Internal Server Error

Looking at HTTP traffic

To see HTTP requests and responses

- in Firefox, using

  `Tools -> Web Developer -> Network`

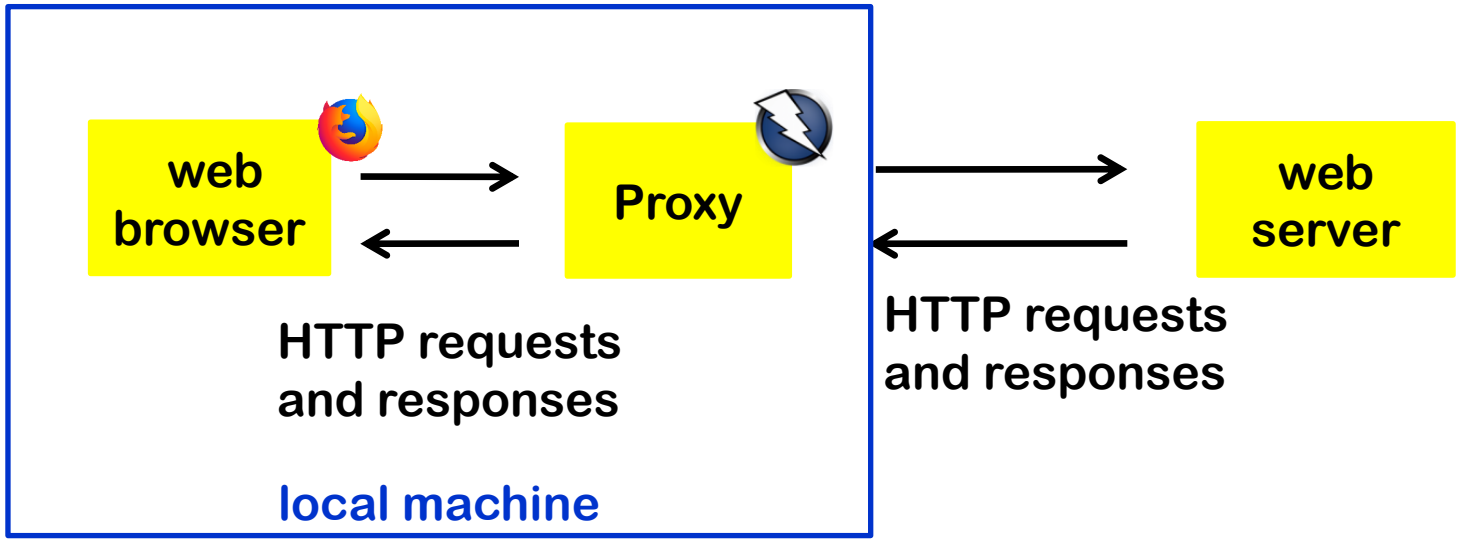
  or

  `CTRL-SHIFT-E`

- using a tool that acts as a proxy
  - OWASP ZAP (Zed Attack Proxy)

Recordings of short demo in Brightspace Virtual Classroom!

Proxy



Proxy can observe – and alter – any incoming or outgoing traffic.

HTML (Hypertext Markup Language)

The body of an HTTP response typically consists of HTML

HTML combines

- data: content and markup, eg `<b> .. </b>` for bold text
- code: client-side scripting languages such as JavaScript

and can include tags for (pointers to) content from other web sites, eg

- `<a href ..>` to add clickable link
- `<img ..>` to include an image
- `<script ..>` to include a script

The 'living' HTML standard, updated 25 Jan 2024, is > 1000 pages.
See https://html.spec.whatwg.org

Looking at HTML

- You can view the raw HTML in your web browser

  Eg in Firefox, using `View -> Page Source`

  Try this, if you have never done this.

HTTP: GET and POST

Two HTTP request methods:

- GET: used to retrieve data

  For example, retrieve an HTML file

- POST: used to submit a request and retrieve an answer

  For example, order a plane ticket

GET should be used for idempotent operations, ie. operations without side effects on the server, so that repeating them is harmless

The term comes from mathematics: f is idempotent iff $f(f(x)) = f(x)$

E.g. rounding or taking the absolute value of a number are idempotent operations, squaring is not.

GET vs POST

Parameters (aka query strings) treated differently for GET and POST

- GET: parameters passed *in URL*

```
www.ru.nl/login_form.php?name=erik&passwd=secret
```

- POST: parameters passed *in the body* of the HTTP request

```
POST www.bla.com/login_form.php
Host www.ru.nl
name=erik&passwd=secret
```

GET vs POST

GET has parameters in URL

POST has parameters in body

An attacker observing the network traffic can see parameters of both GET and POST requests. Still, there are differences:

GET requests

- can be cached
- can be bookmarked
- end up in browser history
- hence: should not be used for sensitive data!
- have a maximum length

POST requests

- are never cached
- cannot be bookmarked
- do not end up in browser history
- have no restrictions on length

forms in HTML

Forms in HTML allow user to pass parameters (aka query string) in an HTTP request as GET or POST

```
<form method="GET" action= "http://ru.nl/register.php">
    Name: <input type="text" name="First name">
    Email: <input type="text" name="Last name">
     <input type="submit" value="Submit">
</form>
```

Please enter your name to register

First name: [ ] Last name: [ ] Submit

See http://www.cs.ru.nl/~erikpoll/websec/demo/demo_get_post.html

example HTTP response

```
HTTP/1.1 200 OK
Date: Fri, 11 Apr 2014 14:07:12 GMT
Server: Zope/(2.13.10, python 2.6.7, linux2) ...
Content-Language: nl
Expires: Tue, 11 Sep 2014 14:07:12 GMT
Cache-Control: max-age=0, must-revalidate, private
Content-Type: text/html;charset=utf-8
Content-Length: 5687
Set-Cookie: keyword=value,...
<HTML>
....
</HTML>
```

NB information leakage about web *server* used. Potentially useful for attacker!

example HTTP request

```
GET /oii/ HTTP/1.1
Host: www.ru.nl
Connection: keep-alive
User-Agent: Mozilla/5.0 ... Firefox/3.5.9
Accept: text/html,application/xml...
Referer: http://www.ru.nl/
Accept-Encoding: gzip,deflate
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Cookie: keyword=value...
```

NB information leakage about *browser* used. Potentially useful for attacker!

For you to do

- Read part I & II of "Understanding The Web Security Model"

- Check out the demos
  - http://www.cs.ru.nl/~erikpoll/websec/demo/demo_get_post.html
  - http://www.cs.ru.nl/~erikpoll/websec/demo/demo_javascript.html
  - http://www.cs.ru.nl/~erikpoll/websec/demo/demo_DOM.html

Exercises for tomorrow:

A. Install WebGoat and ZAP proxy

B. Try out ZAP by doing the exercises mentioned in http://www.cs.ru.nl/~erikpoll/websec/demo/demo_get_post.html

C. Do the WebGoat exercises for the coming week

More info on all this in Brightspace