Web Security

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 rapidly evolving

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

Goals of this course:

- How do attacks on the web work?
- What we can do about them?
- Why are these attacks possible?

Wider context

Most security problems arise from attacks on

1. PEOPLE
2. SOFTWARE
3. interaction & misunderstandings between people & software

Common attacks on software are

- attacks exploiting memory corruption (treated in Hacking in C)
- attacks on web technology (this course)

Organisation

Weekly lecture

- read the slides & any reading material mentioned
- try out the demo webpages mentioned in the lecture
- ask questions in Discord channel for the lecture

Weekly lab session with 3 types of exercises

1. lessons on OWASP WebGoat
   - no need to hand these in
2. challenges at `http://websecurity.cs.ru.nl`
   - handed in automatically when you complete them

   NB for this you will need your Science login
3. ad-hoc assignments
   - to be handed in via Brightspace

Help with lab sessions on Wednesdays 12:30-15:15 via Discord

- Work in pairs
- Doing the exercises is obligatory to take part in the exam

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

Course materials

All info & course material is in Brightspace

Obligatory reading

- all the slides
- some articles & blog posts linked to in Brightspace
- ‘Surviving the Web: A Journey into Web Session Security’
  Stefano Calzavara et al. (ACM Computing Surveys, Vol. 50 No 1, 2017)

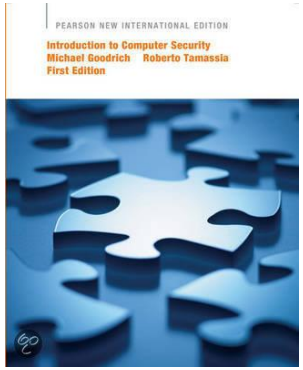
Optional background reading:

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, 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 team at ctf-ru.slack.com

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

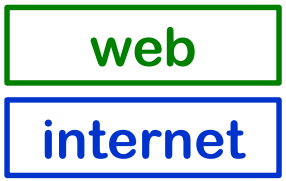
The internet & the web

The internet & The web

Often confused, but they are different

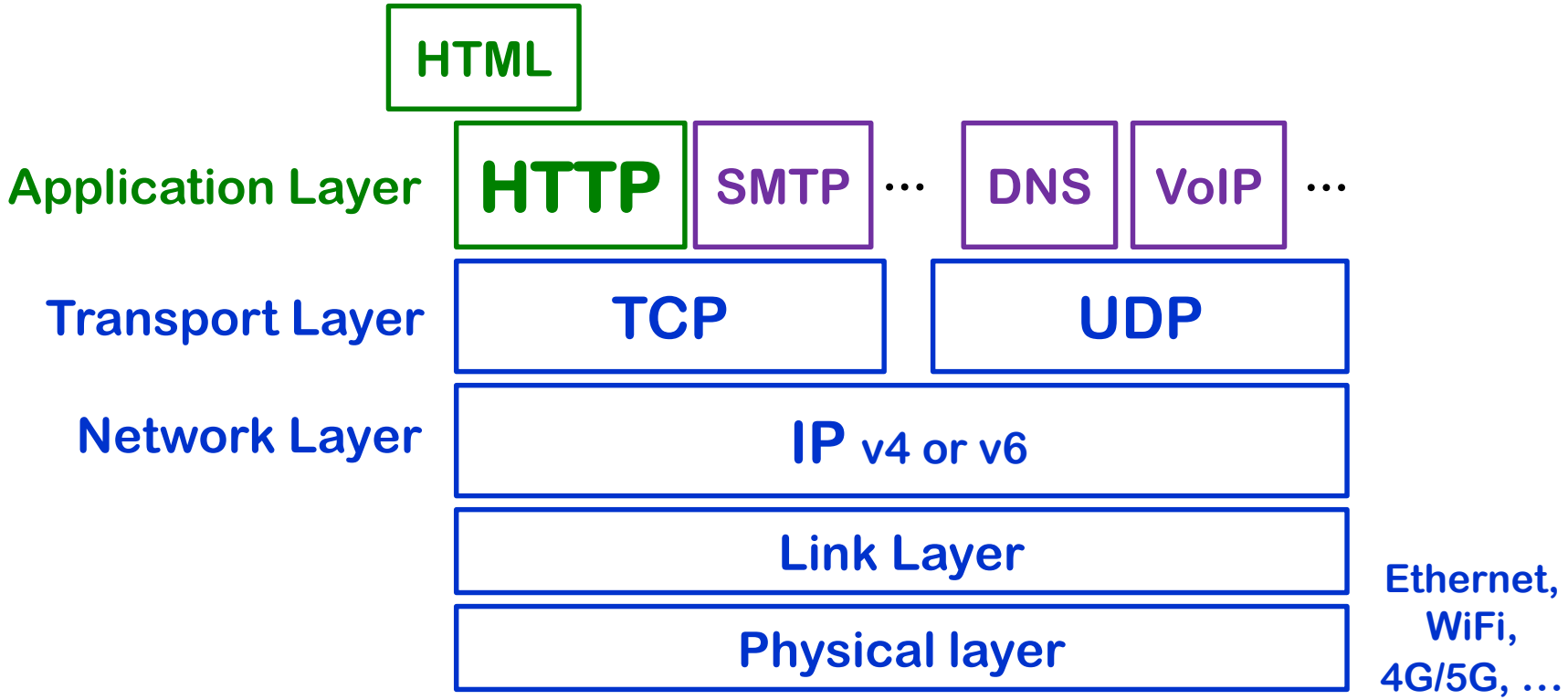
- **The internet**
  - **provides networking between computers**
  - **using the IP protocol family with UDP or TCP**

- **The web**
  - **collection of services that can run *over* the internet**
  - **using the HTTP/HTML protocol family**



The internet & The web

- Protocol stack of many languages and protocols
- Various services can be provided over the internet: email (SMTP), VoIP, ftp, telnet, ssh, … and HTTP



The world wide web

The web is one of the services available over the internet

www = HTTP + HTML + URLs

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 response

At the client side, it involves a web browser

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 answer the phone, when they buy a coffee,…

Protocols usually specify two aspects of interaction:

1. language / data format for messages
   - e.g. specified by regular expression or grammar

2. correct / expected *sequences* of messages
   - e.g. specified by finite automaton aka state machine or a Message Sequence Chart (MSC)

Aside: Languages (or formats)

For example

- file formats: .html, .docx, .pdf, .txt, .mp3, .jpeg, .mp4, .js, …
- 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?

Complexity and ambiguity in languages are major root causes of security problems

Evolution of the web

Evolution of the web

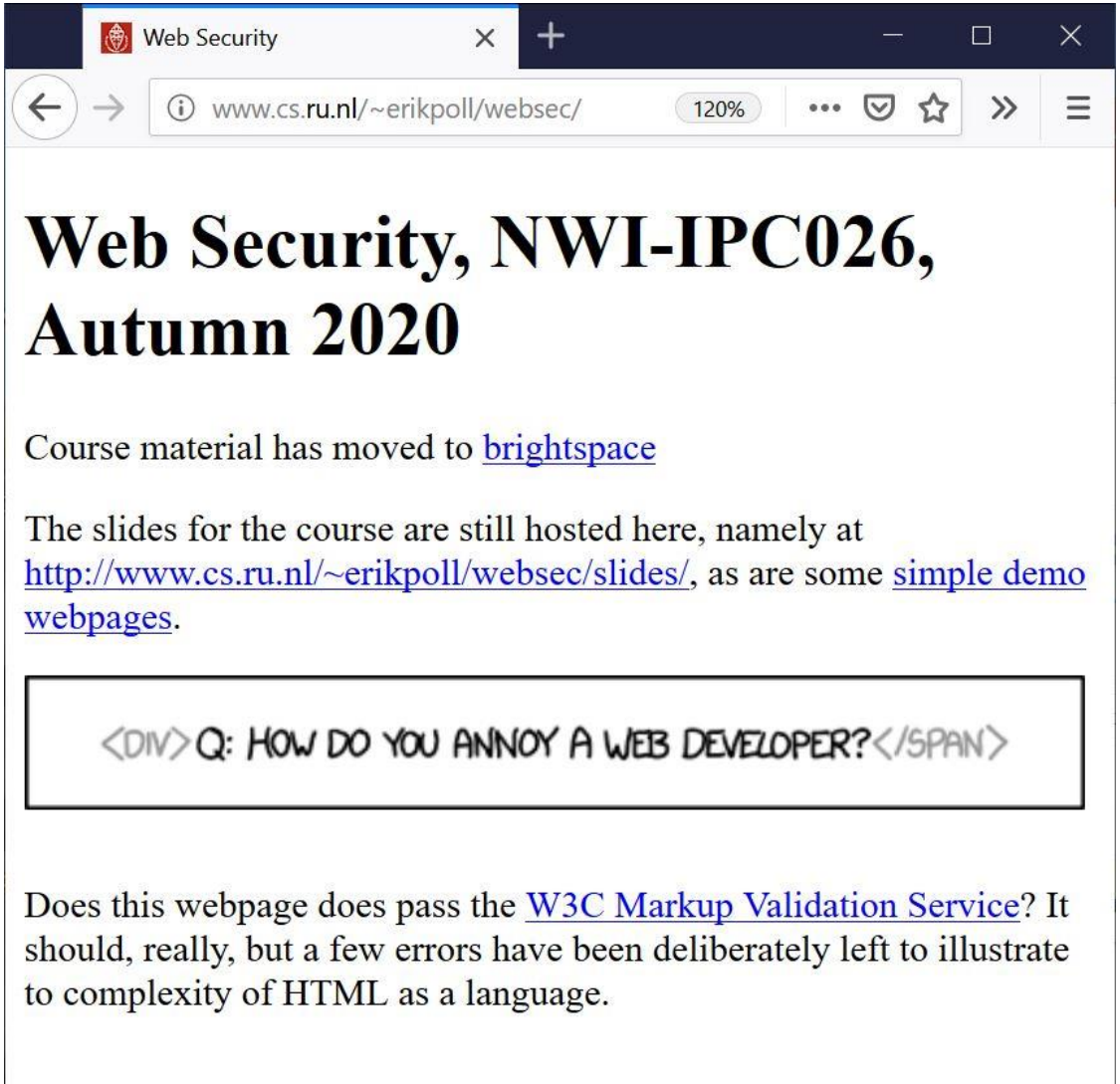
Web is constantly evolving

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

1. Static hypertext
2. *Dynamically generated* web pages
   - Web 2.0
3. *Dynamic* web pages
   - aka web apps
4. Ajax: asynchronous interaction between browser & server
5. More Web APIs
6. Apps on mobile phones & tablets

1. Static hypertext

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



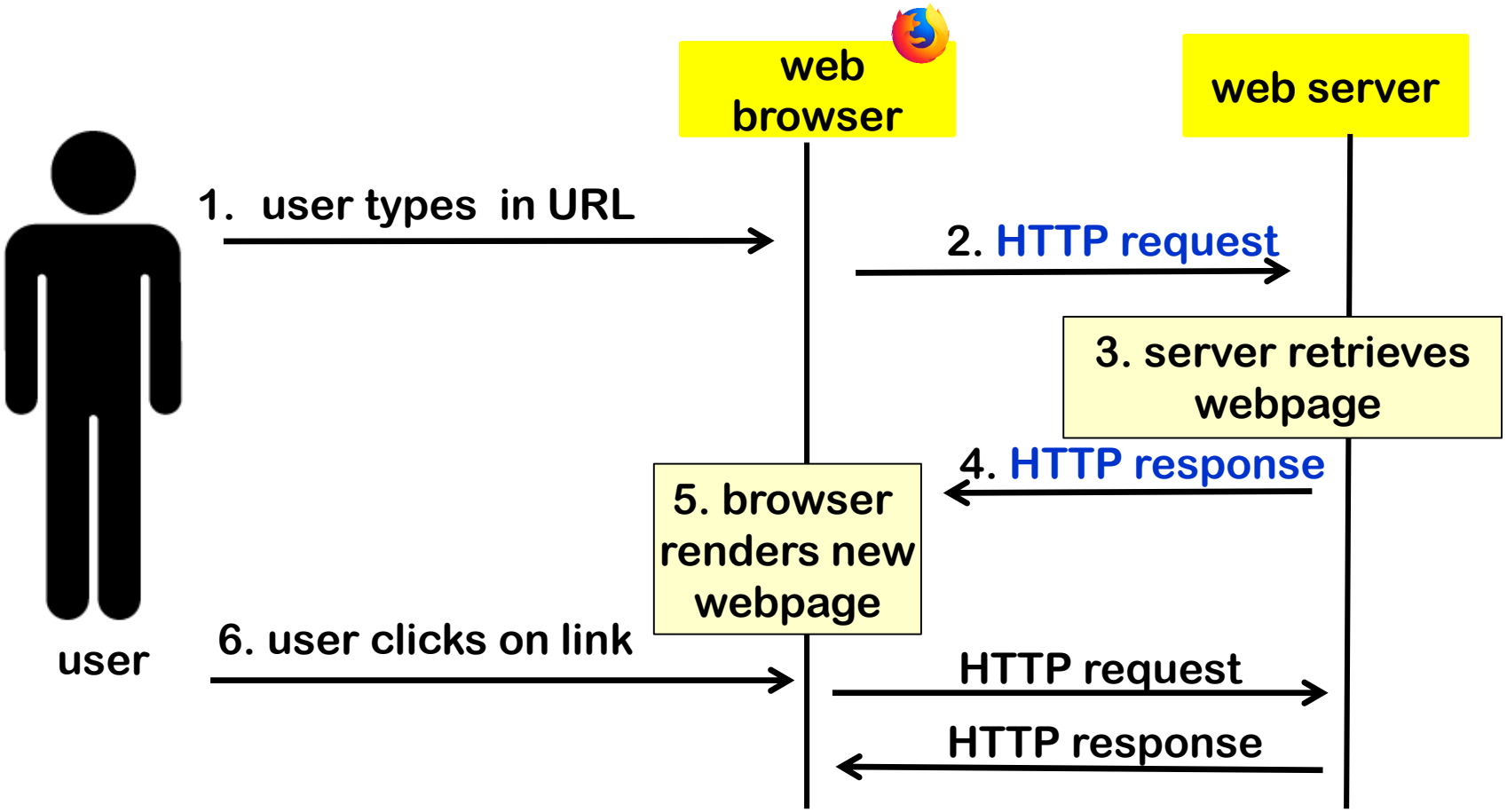
1. Static hypertext

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

- Content of such a webpage can simply be a fixed file on the file system, so a (very simple) web server only has to retrieve files from disk
- The content doesn't 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

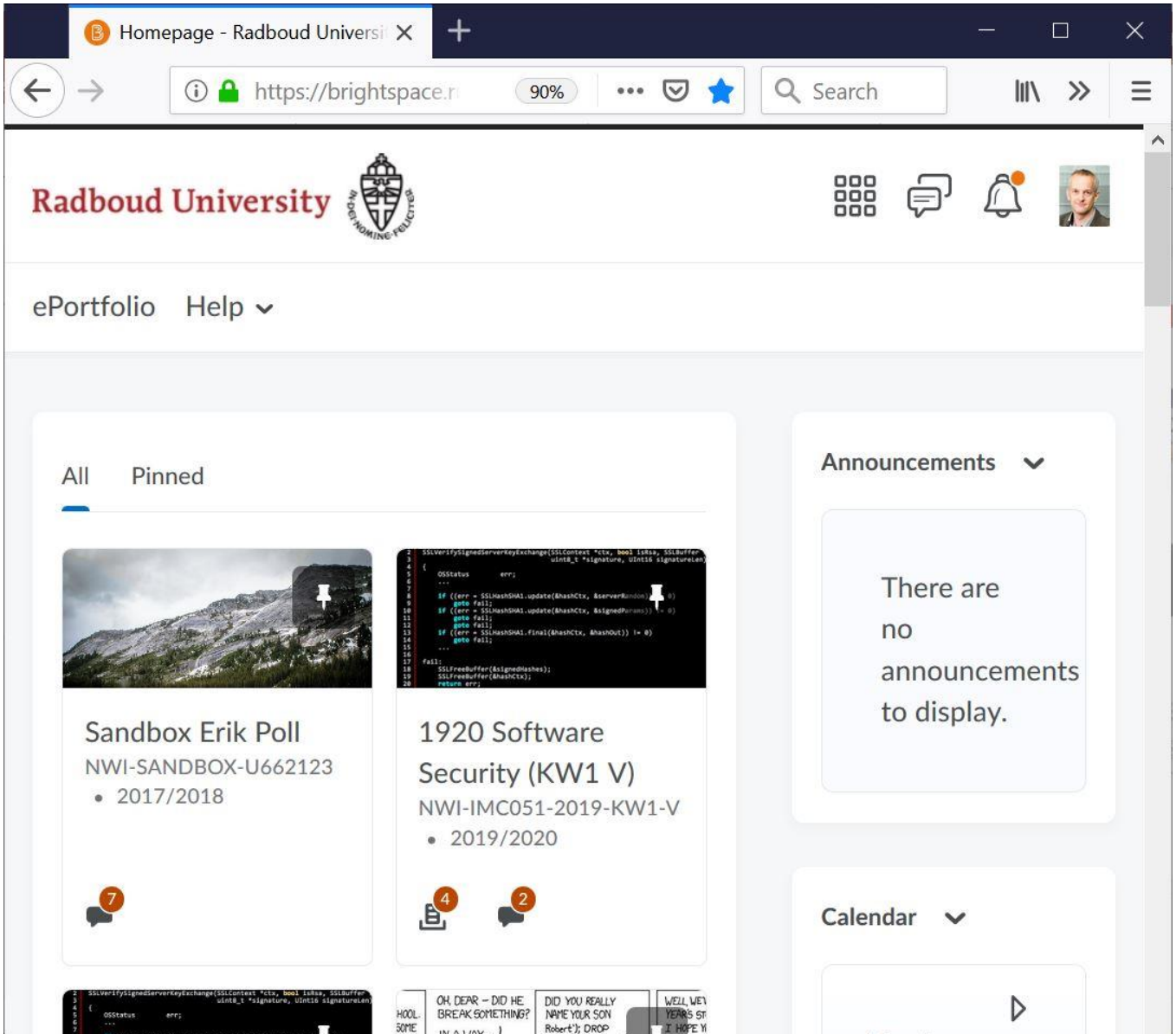
Eg `http://www.cs.ru.nl/~erikpoll/websec/index.html`

Synchronous interaction on the web

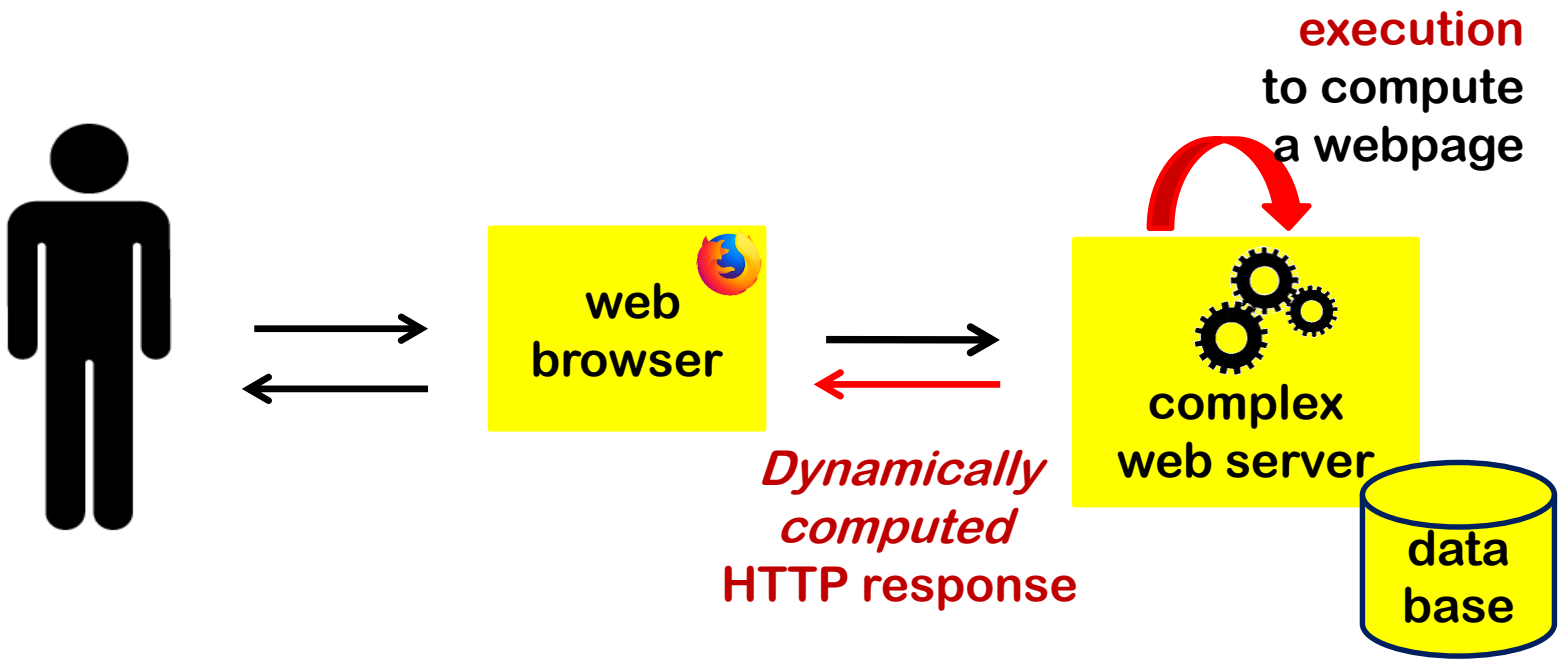


This is overly simplistic. Even very simple browsing is much more asynchronous. E.g. browser will start rendering while images are retrieved.

2. *Dynamically created* web pages



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

Web page is dynamically created, on demand

Eg google, gmail, facebook, brightspace, amazon, …

Different users will be served a different webpage

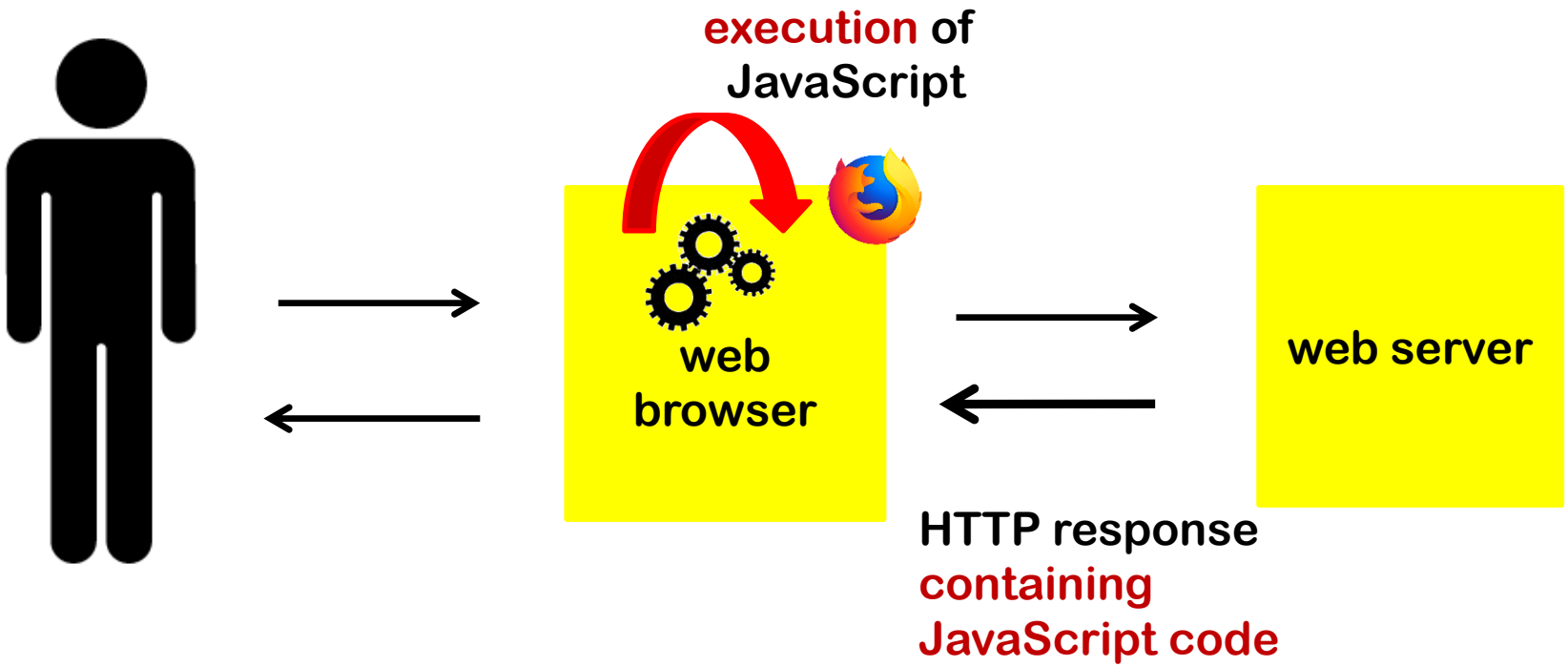
The web server now runs a web application

- The web applications run in a web application server
  - eg Apache Tomcat, Websphere,…
- The applications are written in scripting or programming languages
  - eg CGI, Perl, Python, PHP, Java, C#, Ruby on Rails, Go, …

This allowed web 2.0, with user-generated content

in web forums, Wikipedia, and social media: facebook, Instagram, twitter,…

3. Dynamic web pages



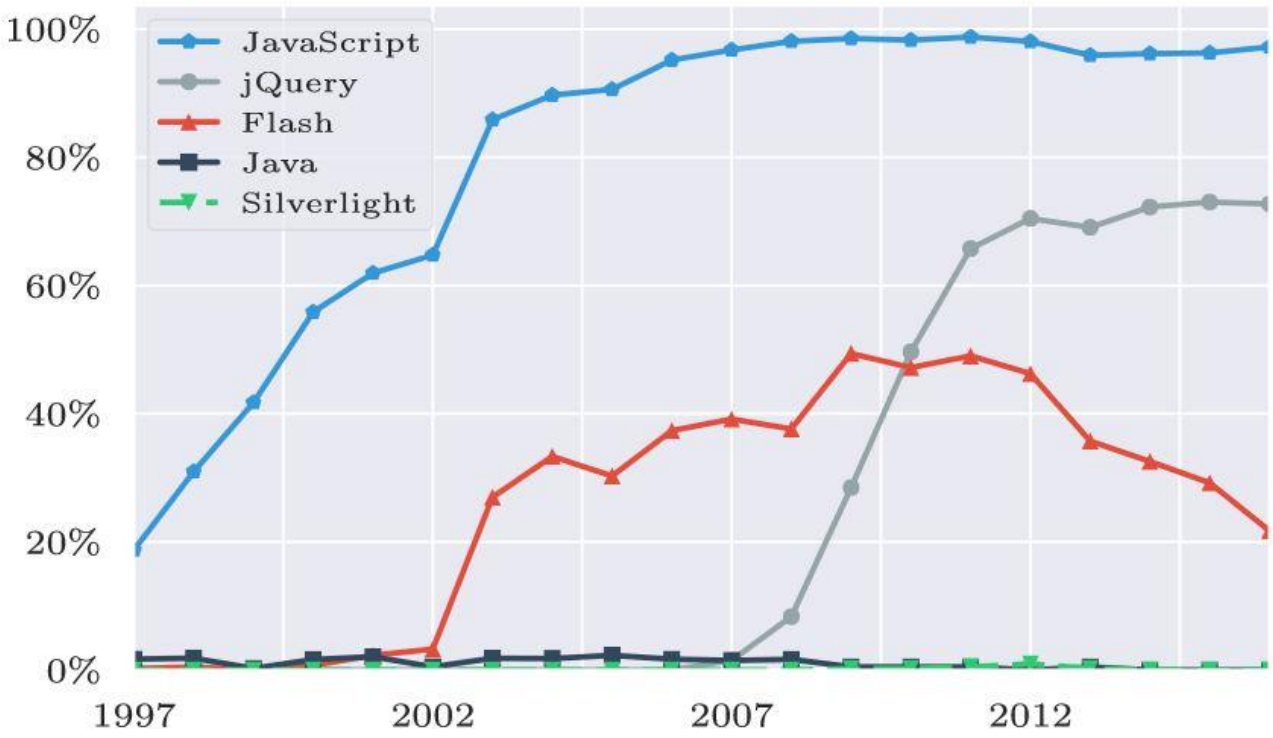
Eg. http://www.cs.ru.nl/~erikpoll/websec/demo/demo_javascript.html

3. *Dynamic* web pages

- Web pages include code that is executed *in the browser*
- Two main languages for this:
  - JavaScript
    - part of the HTML5 standard
  - WebAssembly (Wasm)
    - since 2017
- Older languages used for dynamic behavior in the browser included Java, ActiveX, Flash, Silverlight, …

- Goals:
  - more attractive web pages
  - more and faster interaction with the users
    - there can be interaction between the user & browser, and changes to the webpage, *without a new page being loaded*

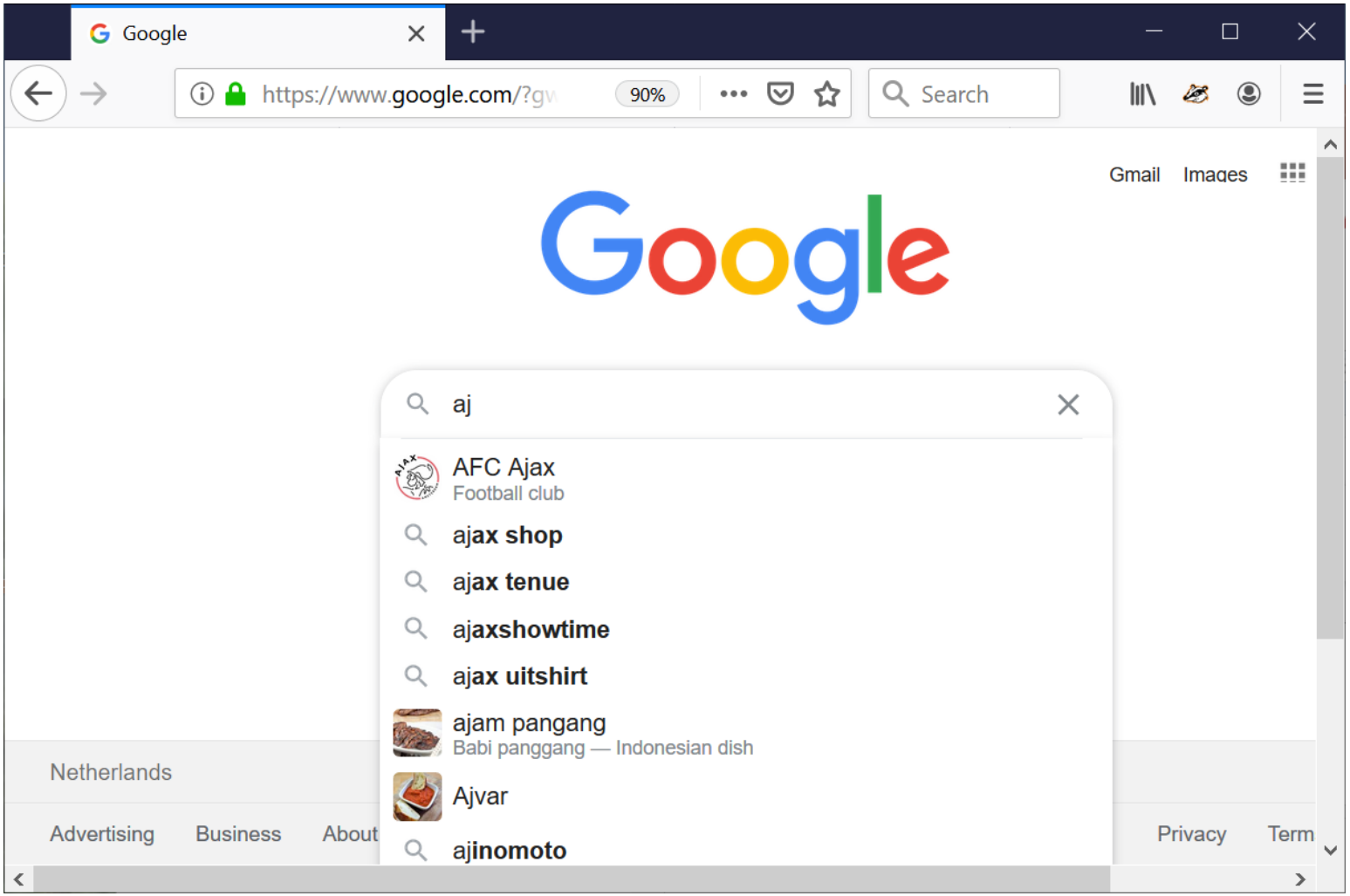
Evolution in web technologies



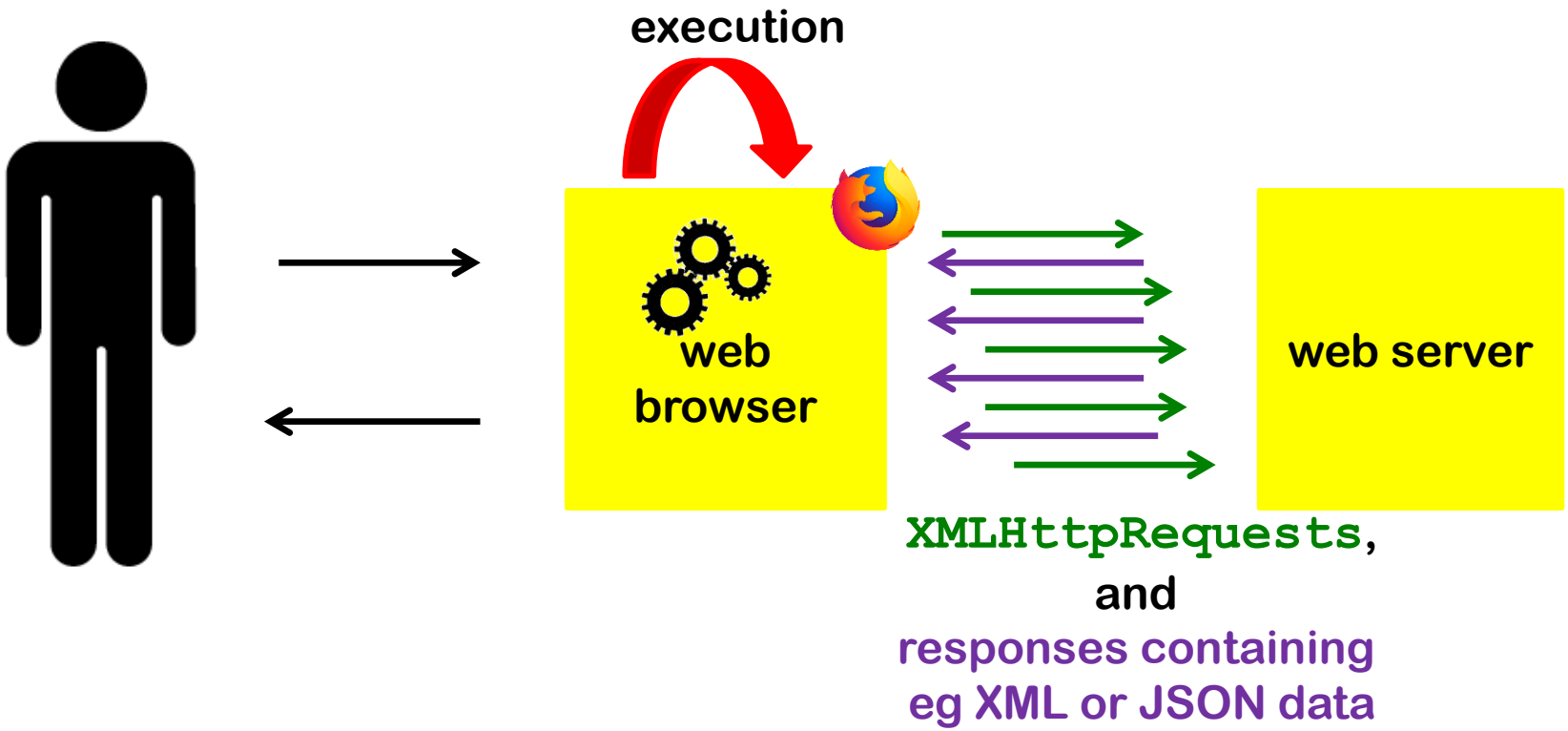
Technologies used by top 500 web sites

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

4. asynchronous interaction with Ajax



asynchronous interaction with Ajax

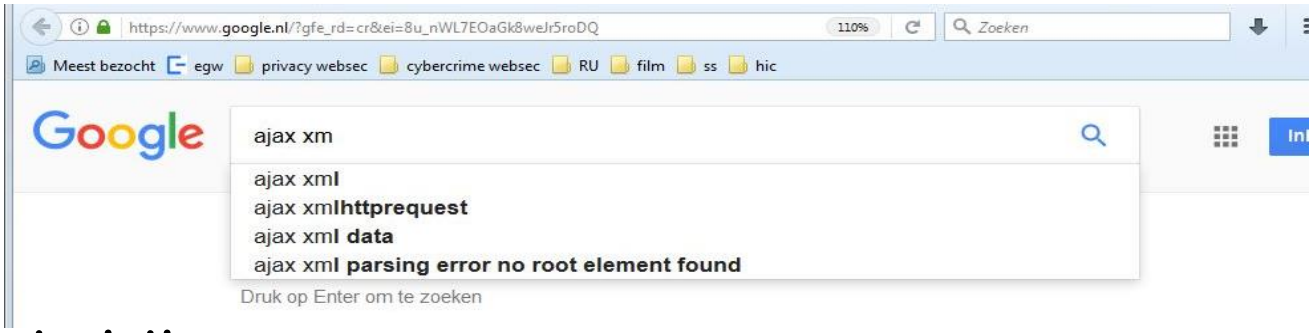


With Ajax the initiative for interaction still lies with the browser;
With `WebSockets` communication becomes full duplex
ie. web server can take initiative to send message

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: code can update *part* of webpage

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

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
- XML has support for schemas (i.e. definitions of XML 'dialects'), but there now a draft spec for JSON schemas

HTML vs XML (& JSON)

- HTML is fixed and only defines how information should be displayed

  eg `<b>Display this text in bold</b>`

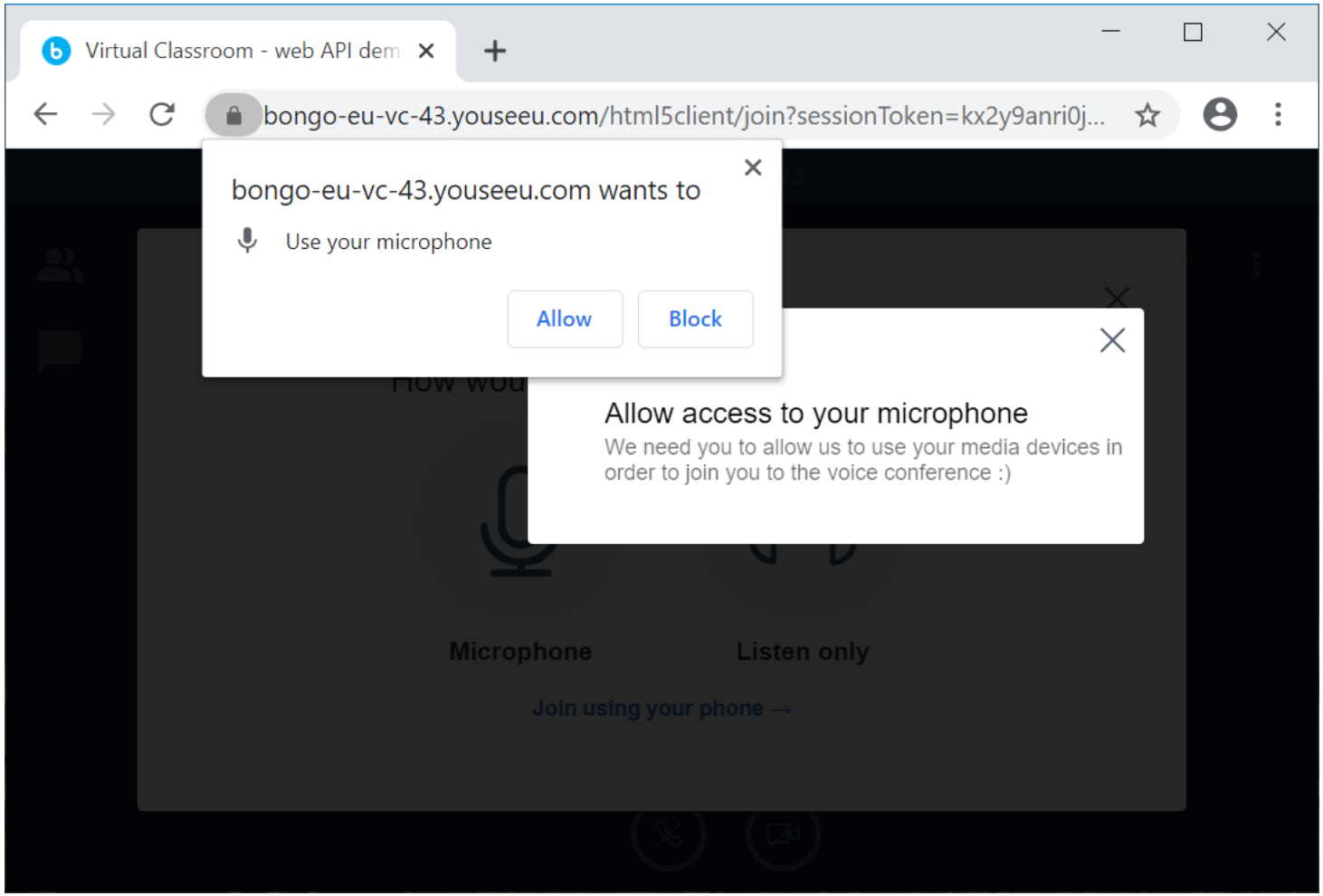
- XML is extensible and carries semantic information in tags, ie what it means

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

Some people hoped for a Semantic Web, aka Web 3.0, where all data would have such meaningful tags, to facilitate automated processing

- eg web scraping would become a lot easier

5. More Web APIs in browsers



5. More Web APIs

Via Web APIs the browser provides functionality to web pages (and JavaScript of Web Assembly *in* web pages)

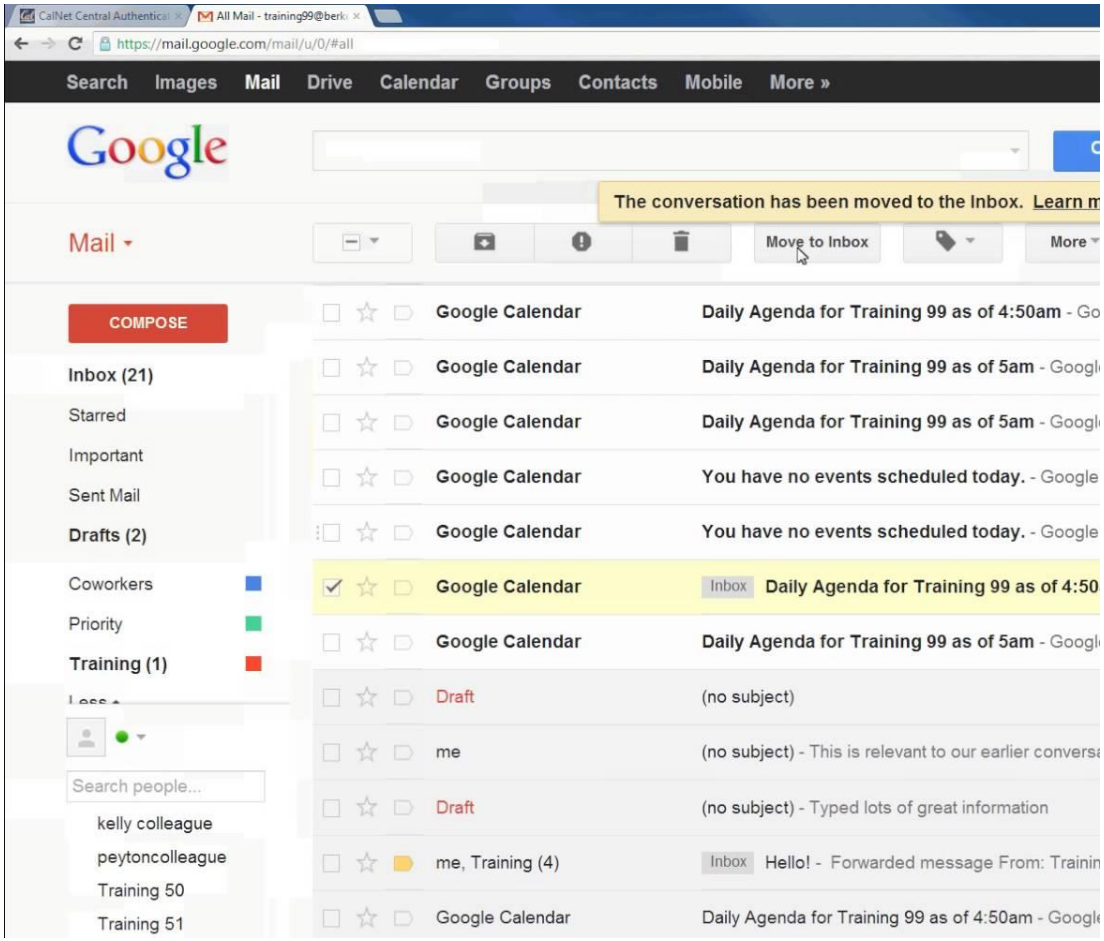
The set of Web APIs is constantly evolving, with some differences between browsers.

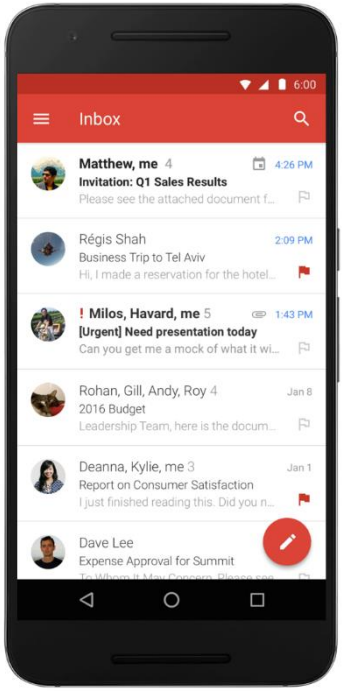
- **Many Web APIs have been added over the years: for sound, accessing web cam, microphone, allowing screen sharing, using local storage on the computer, …**

- **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 browser to apps





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
Many apps use an HTML rendering engine, eg WebKit, as used in browsers.

Some apps are simply stand-alone dedicated browsers that display HTML contents. (Some of this HTML content can be pre-loaded in the app, and not retrieved over the web, for fast start-up.)

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

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.

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

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

NB there are many RFCs, and they can be quite complex!

Eg. look up the definition of an email address in RFCs 5321, 5322, 3696 (with errata in http://www.rfc-editor.org/errata_search.php?rfc=3696) and RFC 6531 for the international character extensions.

URLs

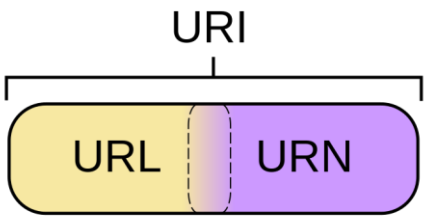
scheme://login:password@address:port/path/to/resource?query_string#fragment

1 2 3 4 5 6 7

1. scheme/protocol name, eg http, https, ftp, file, ...
2. credentials: username and password (optional)
3. address: domain name or IP address
4. port: port number on the server (optional)
5. hierarchical path to the resource
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.

URI vs URL



Lots of confusion about the correct terminology

- URL: Uniform Resource Locator
- URI: Uniform Resource Identifier

In most discussions about the web, these are effectively synonyms.

I will only use the term URL in this course
but strictly (pedantically) speaking, a URL is a special kind of URI

URIs that are not URLs: URNs (Uniform Resource Names),
that specify a *name* of a resource, but not a *location* where to find it.

Classical example: ISBN 12920254909, which identifies a unique book,
but not where to find it, so it's a URN but not a URL

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`

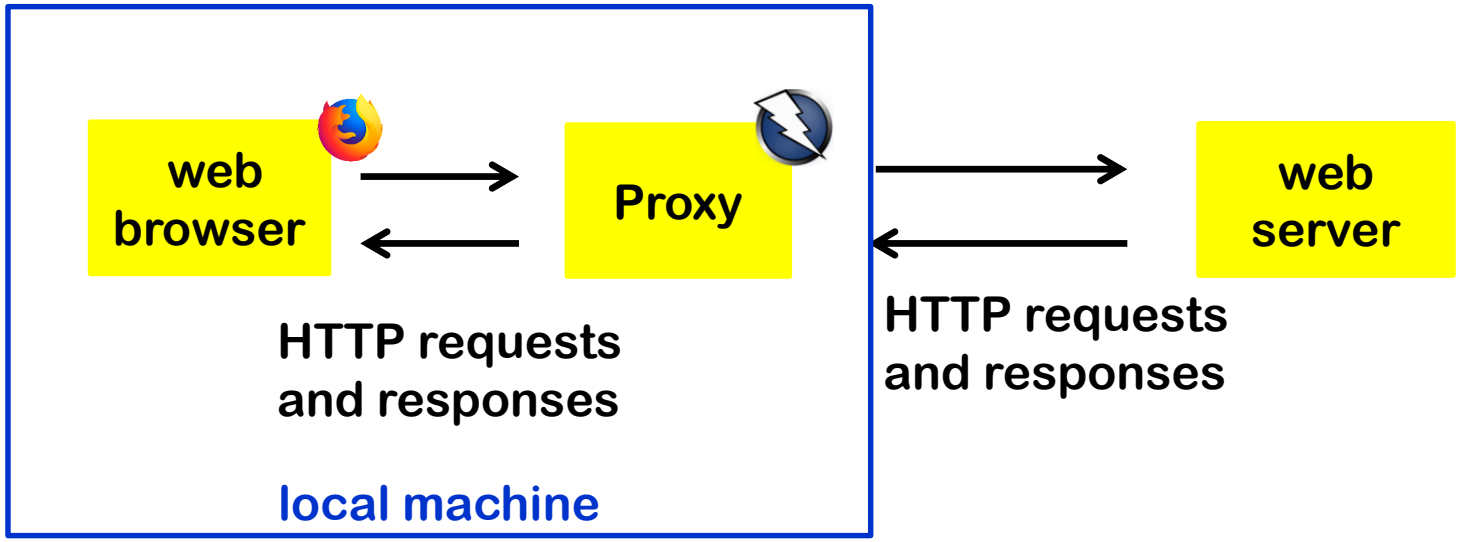
  or

  `CTRL-SHIFT-E`

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

Recordings of short demos 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 latest spec of HTML, version 5.2, updated 30 Aug 2020, is 1297 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.

COMPLEXITY in browser: many nested languages & formats

```
<html>
  <img scr="http://www.ru.nl/logo.jpg">

  <a href="https://duckduckgo.com/?q=is+/+special%3F">

  <script> var x = 'string';
             // a JavaScript program
  </script>
</html>
```

- Double quotes in `<img>` moves to URL context.
- The URL consists of different parts:
  eg.the query string after the ?, where / is no longer a reserved character
- The `<script>` tag moves from HTML to JavaScript context.
- The single quote inside JavaScript moves to JavaScript string context.

URL encoding

Replaces reserved characters that have a special meaning in URLs

`/?!*';:@&=+$,#()[]`

with their ASCI value in hex preceded with escape character `%`

/	#	space	=	?	%	...
`%27`	`%23`	`%20` or `+`	`%3D`	`%3F`	`%25`	...

Try this out with eg `https://duckduckgo.com/?q=%3F`

Possible sources of confusion (and bugs or security issues?)

- Encoding space as + comes from older x-www-form-urlencoded format
- The reserved characters are different for different parts of the URL. Eg / in the path of a URL must be encoded, in the query it need not be
- What happens if you URL-encode unreserved characters? eg `A -> %42`
- What happens if you double URL-encode? eg `% -> %25 -> %2525`

HTML encoding

Replaces HTML special characters with similar looking ones

<	>	&	“
<	>	&	"

- HTML encoding and URL encoding are *very* different things, used for *very* different contexts
  - *still, things can get confusing: what about URLs inside HTML? what about javascript inside HTML?*
- HTML also has the notion of character encoding: which character set is used, eg ASCI or UTF-8 (default)
- Some browsers are sloppy/forgiving, and will let you get away with *not* encoding `&` as `&` in webpages
  - http://validator.w3.org checks if a page is correct HTML
- On top of HTML-encoding, websites may apply additional input sanitisation to remove or replace tags it wants to disallow in user input;
  - eg `<script>` tags are commonly stripped from user input

base64 encoding

HTTP is text based, so all data transmitted has to be text
– ie. printable, displayable characters

Base64 encoding turns 'raw' binary data – ie bytes into text so that it can be transferred via HTTP

- 6 bits coded up as one of the standard characters
  `a-z A-Z 0-9 + /`
- So 3 bytes represented as 4 characters
- Padding with = or == to make sure results is multiple of 4 characters long

base64 encoding

HTTP is text based, so all data transmitted has to be text
– ie. printable, displayable characters

Base64 turns 'raw' binary data – ie bytes into text
so that it can be transferred via HTTP

- using the 64 characters `a-z A-Z 0-9 + /`

Bits		0 1 0 0 1 1	0 1 0 1 1 0	0 0 0 1 0 0	
Base64 encoded	Sextets	19	22	4	Padding
	Character	T	W	E	=
	Octets	84 (0x54)	87 (0x57)	69 (0x45)	61 (0x3D)

base64 encoding

- groups of 6 bits coded up as one of the standard characters

  `a-z A-Z 0-9 + /`
- So 3 bytes represented as 4 characters
- Padding with zeroes to make the input a multiple of 6 bits
- Padding with `=` or `==` to make sure results is multiple of 4 characters long

See also https://en.wikipedia.org/wiki/Base64

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

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.htm
- http://www.cs.ru.nl/~erikpoll/websec/demo/demo_DOM.html

A. Install WebGoat and ZAP proxy

B. Try out ZAP

by looking at HTTP traffic generated by
http://www.cs.ru.nl/~erikpoll/websec/demo/demo_get_post.html

- check if parameters end up in URL or body for GET and POST

C. Do the WebGoat exercises for the coming week