Software and Web Security

deel 2

Software and Web Security - 1 & 2

Software is the main source of security vulnerabilities
esp in systems accessible via a network

- part 1
  security problems in machine code,
  compiled from C(++) usually,
  running on standard CPUs

- part 2
  security problems on the web/in software for the web, incl.
  in web browsers (at client side) and
  in web applications and web application servers (at server side)
  and the *interaction between them*

Prerequisites

- Security
  - esp. notions of confidentiality, integrity, and availability (CIA) and authentication
- Databases and Security
  - esp. SQL, since database in a web-application is an important target for attacker
- (Software and Web Security 1)
  - useful to note the recurring patterns and trends in weaknesses and attacks
  - knowledge of C(++) and associated memory weaknesses is NOT required
  - read through the slides of first lecture of this course, if you did not take this course

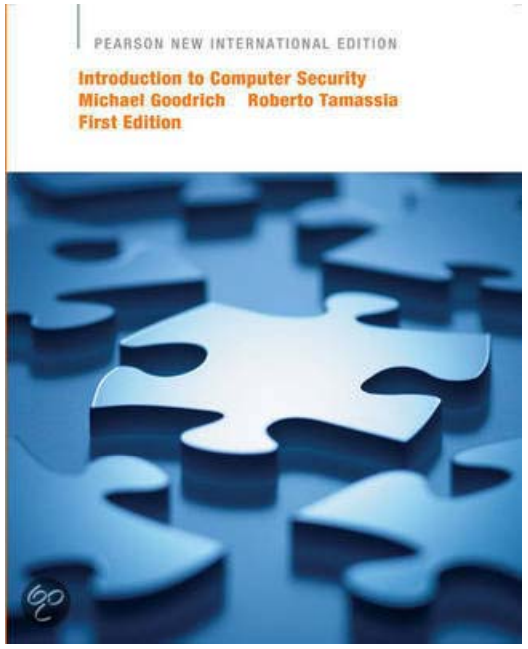
Course materials

Book: Introduction to Computer Security

by Michael Goodrich and Roberto Tamassia

(Pearson new international edition 2013/2014)

Chapters 1, 5.1, 7



Additional info & course material on

http://www.cs.ru.nl/~erikpoll/sws2

Lab exercises using OWASP WebGoat

Weekly lab session with web hacking exercises using WebGoat

- Tuesdays 8:45-10:30 in terminal room HG00.075

  Ko Stoffelen and/or Willem Burgers will be there to help
- Exercises to be done in pairs
- Doing the exercises is **obligatory** to take part in the exam
- More importantly, exam questions will assume familiarity with the lab exercises

Recap terminology and concepts from Software & Web Security 1

weaknesses vs vulnerabilities

Common use of terminology can be very confused & confusing:

(security) weaknesses, flaws, vulnerabilities, bugs, errors, coding defects, ...

We can make a distinction between

- security weakness/flaw:

  something that is wrong or could be better, and
  which *might* become a security problem

- security vulnerability

  a weakness/flaw that *can* actually be exploited by an attacker

  This requires the weakness to be

  1. accessible - attacker has to be able to get at it
  2. exploitable - attacker has to be able to do some damage with it

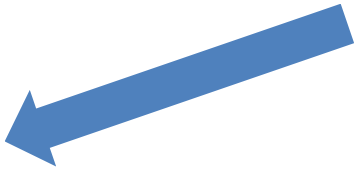
Eg by unplugging your network connection, many vulnerabilities become flaws

design vs implementation flaws

Software vulnerabilities can be introduced at different “levels”

- design flaws
  - fundamental error in the design
- implementation flaws or coding error
  - introduced when implementing

The precise border is not precise



focus of this course

Vulnerabilities can also arise on higher levels (out of scope for this course)

- configuration flaw when installing software on a machine
- the user
- unforeseen consequence of the *intended* functionality (eg. spam)

errors in program logic vs low-level coding defects

We can distinguish

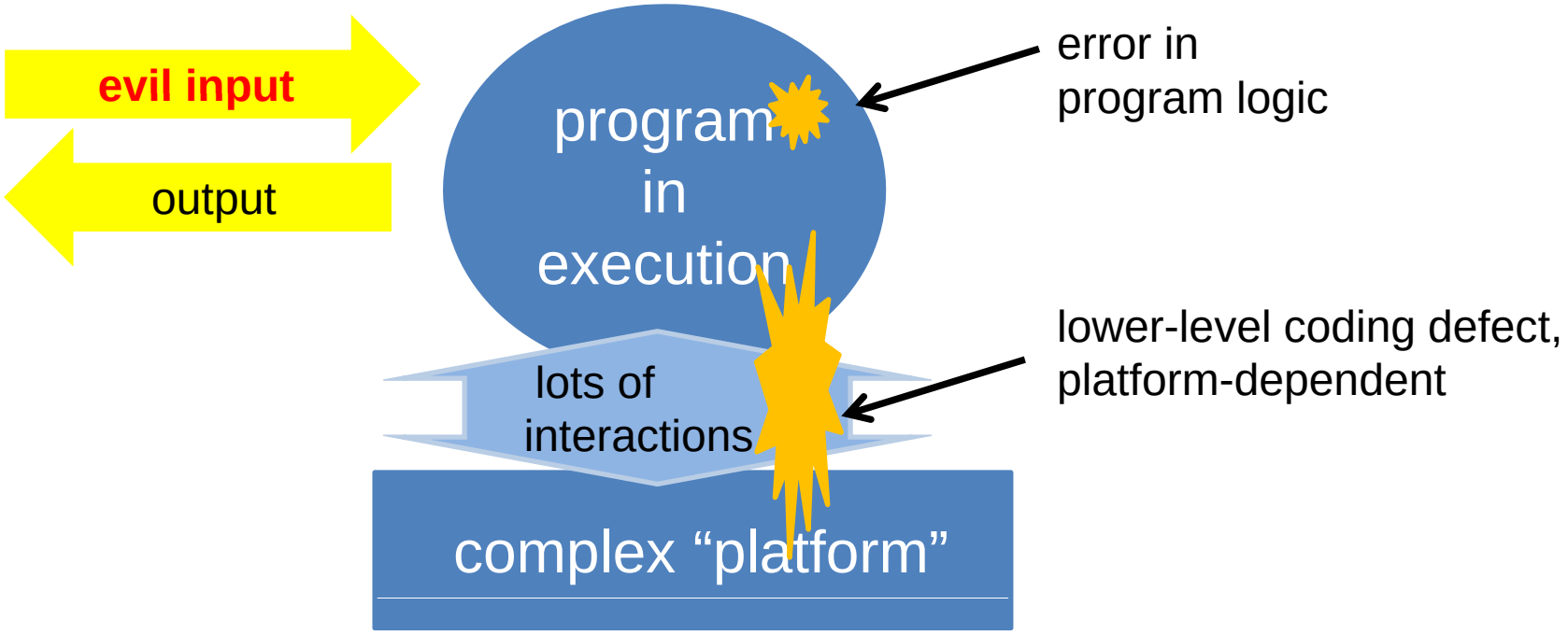
1. flaws that can be spotted just by looking at the program itself (and understanding what it is meant to do!)
   - eg. incorrectly nested if-statements

   Sometimes called logic errors, as opposed to syntax errors,

   or an error in the program logic
2. lower-level, implementation flaws that arise due to interactions with the underlying platform
   - in the case of software running on a normal machine, the platform includes CPU, OS, and memory
   - in the case of software interacting over the web, this platform is "the web" ,incl. a web browser on the client side and a web server or web application on the server side

focus of this course

(malicious/untrusted) **input** as main source of trouble



The platform can be

- the computer (CPU & memory)
- “the web”

Intro

brainstorm: surfing the web

What is the web?

- What do we need to *make* the web?
- What do we need to *use* the web?

brainstorm

- the internet, connecting clients and servers
- client side:

  web browser, possibly with plugins, running on some OS,
- server side:

  web server or

  web application running on web application framework

  incl. back-end database, running on some OS
- specifications to agree on common *protocols* and *formats*
  - IP, HTTP, HTTPS, HTML, DNS, wifi, ethernet, ..
  - URL/URI, IP addresses, email addresses, ...
  - jpg, png, mpeg, mp4, javascript, Flash, ActiveX, Ajax, ...
  - PHP, Java, Ruby,.. or some other scripting/programming language

What is the internet and the web?

The internet and web consist of hardware and software

- hardware
  - network: copper cables, optic cables, ...
  - ICT: servers, routers,...
- software
  - network drivers, browsers, web servers, ...

Alternatively, we can say the internet or the web is simply a set of specifications, that define

- protocols (for communication)
- languages and formats (for data)

that are somehow realised in HW/SW

protocol

A procotol is a

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

Protocols specify *sequences* of *steps*,

in which *data is exchanged* in specific *formats*

Not just between computers; eg. think of protocol that people follow when they answer their phone.

NB systems consisting of interacting/communicating parties are complex, as the number of states grows exponentially with the number of participants.

languages and formats

Defining a language or formats involves

- syntax
  - what are allowed words/sentences/sequences of bytes?
- semantics
  - what do these mean?
  - hence: how should they be interpreted?

internet vs web

- internet
  - provides networking between computers
  - offering the IP protocol family with UDP and TCP

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

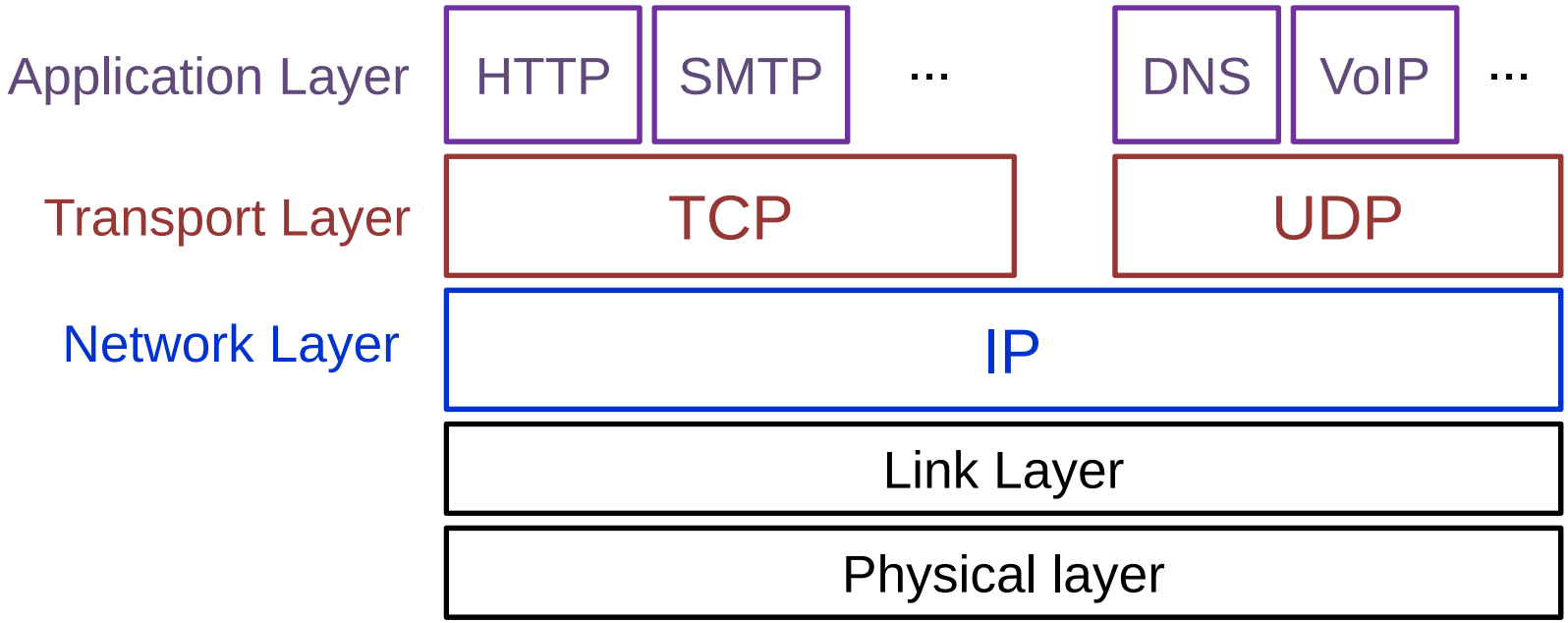
History of
(security worries on)
the internet & the web

Evolution of the Internet

1. the internet
2. the web
3. dynamically generated web pages
4. web 2.0
5. dynamic web pages
6. asynchronously executing content in web pages
7. mobile internet
8. web 3.0 aka the semantic web?

1. internet

- computer network linking computers worldwide
- various services (aka applications) that can be provided over the internet: email, ftp, telnet, ssh, ...
- built using several protocol layers



security worries & problems?

1. The internet is an important attack vector. For any computer connected to the internet, it threatens *CIA of all data and services on it, incl. availability of the computer itself*
2. *Lack of confidentiality, privacy, and anonymity*
   - Any party observing the network traffic (eg router, ISP, ...) can see
     1. the communication exchanged
     2. the fact that two parties are communicating at all

     is seen by any party in the middle, incl. routers, ISP, government agencies, ...
   - Also, parties communicating know each other's IP address

     NB an IP address counts as personal information (persoonsgegeven) in Dutch legislation
3. *Lack of integrity & authentication*
   - when you communicate over the internet, you have no clue who is at the other end of the line, or if information is genuine

integrity and authenticity

Of the trio confidentiality, integrity and availability (CIA),
integrity is the trickiest & most confusing notion.

Usually (always?) by integrity of message
we implicitly also mean authentication of the origin of the message

Conversely, authentication of some party is usually pointless unless you also authenticate (ie. ensure the integrity of) the communication coming from that party.

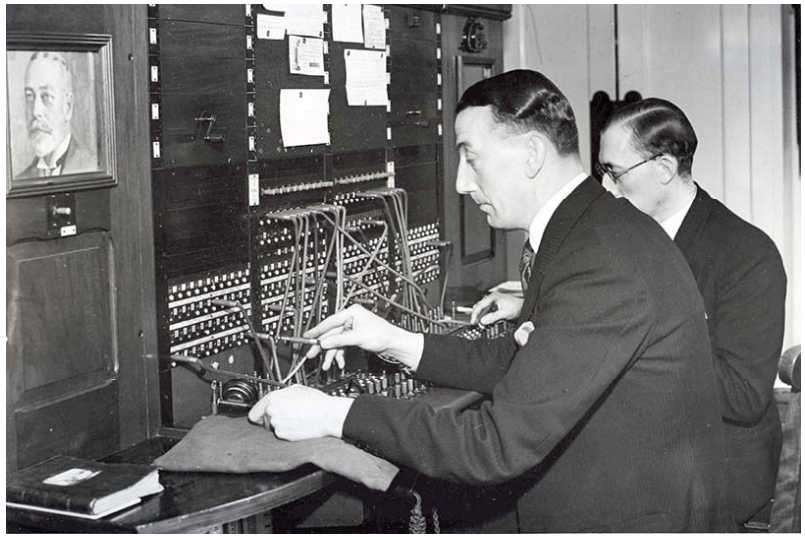
history repeating itself

The telephone network also suffers from poor confidentiality and authentication...

The automatic telephone switchboard was invented by Almon Strowger in 1888. Strowger had an undertaker business and suspected a switchboard operator from transferring calls to him to a competitor.

With automatic switchboard, human operators could no longer do this.

Added benefit for confidentiality:
no need for human operators
that could eavesdrop on calls.



2. the world wide web

The web is one of the services available over the internet

www = internet + HTTP + HTML + URLs

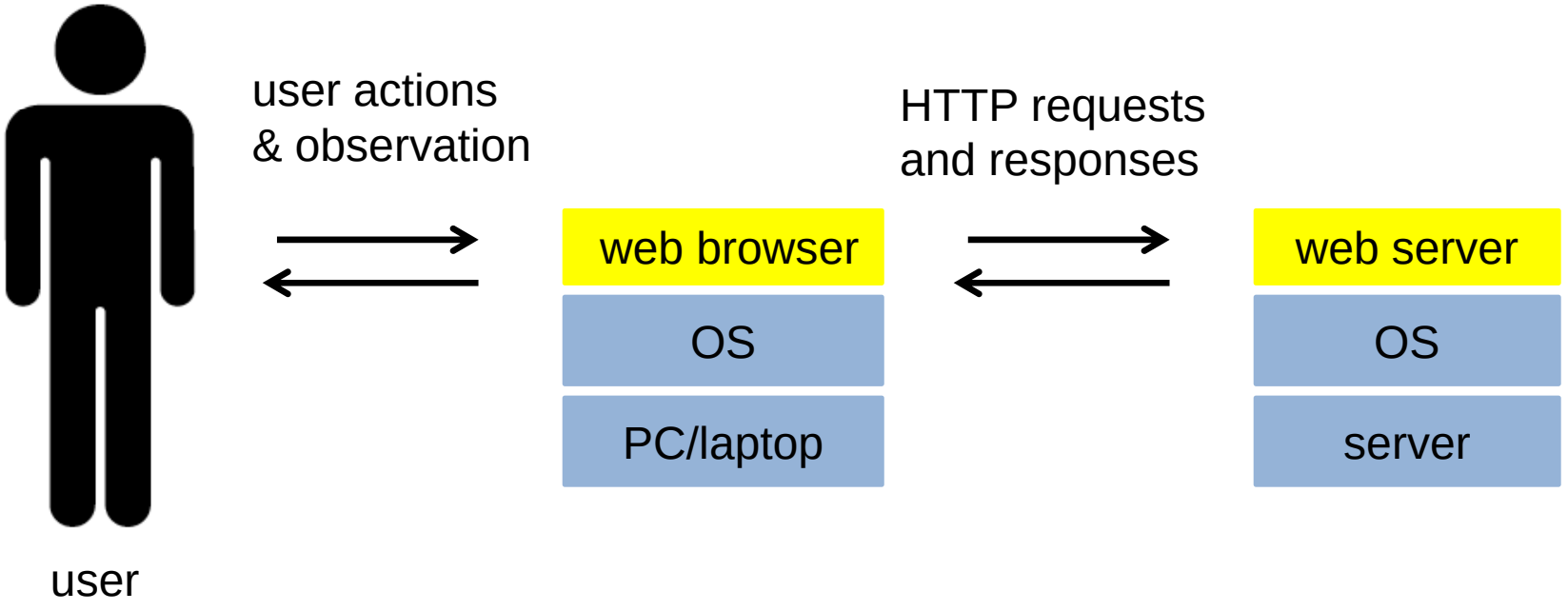
It offers hypertext (text with links) as abstraction layer over some material on the internet.

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

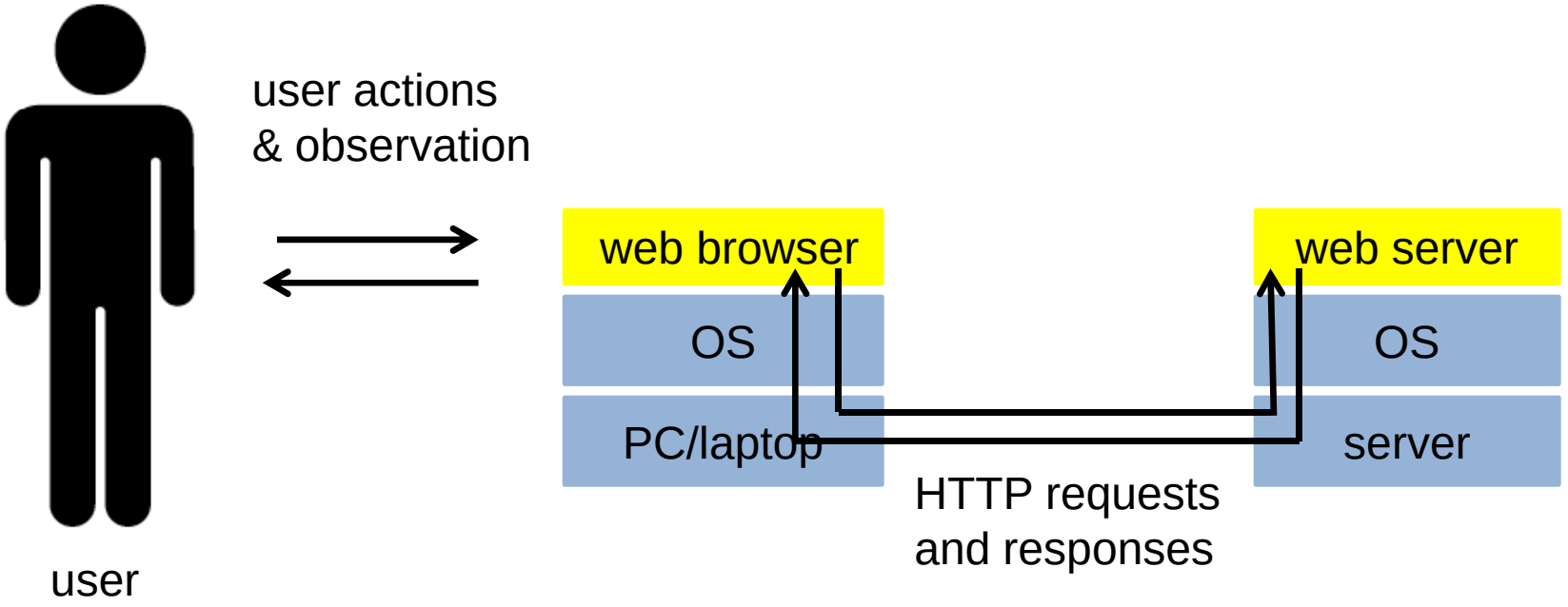
- listens to port 80
- accepts HTTP requests, processes these, and returns some answer

At the client side, it involves a web browser

interactions in surfing the web



interactions in surfing the web



security worries & troubles

The same as for the internet

- *web-connectivity is an important attack vector*
- *lack of confidentiality, privacy, anonymity*
- *lack of authentication and integrity*

security worries & troubles

Additionally,

- *accidentally exposing parts of the file system on the internet*
  Eg **`http://www.cs.ru.nl/~erikpoll/sws2/exam/exam.pdf`**
- *even making this searchable using search engines*

- For most web servers **`.htaccess`** files can be used to configure access to file system.
  Additionally, the OS (Operating System) will do access control.

- Access restrictions for automated web crawlers, as used by search engines, can be specified in **`robots.txt`** files, but it is up to the client to respect these (or not).

3. dynamically created web pages

Simple web pages are static HTML, without any user interaction.

Eg **`http://www.cs.ru.nl/~erikpoll/sws2/index.html`**

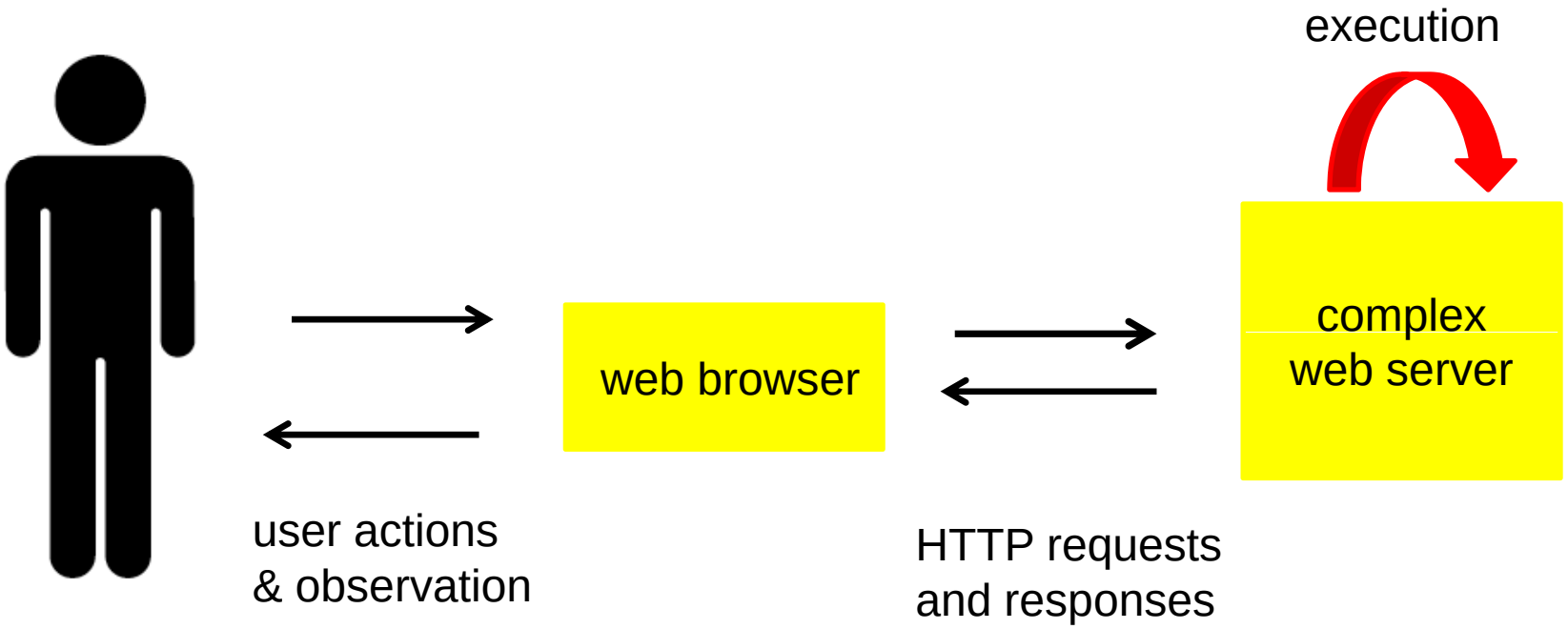
Dynamically created web pages involve a server-side execution engine that dynamically creates web pages on demand

Eg gmail, facebook, blackboard, google,...

The web server now runs a web application.

- These applications run in a web application server
  - eg Apache Tomcat or Websphere
- They are written in scripting or programming languages
  - eg CGI, Perl, Python, PHP, ASP, JSP

dynamically created webpages



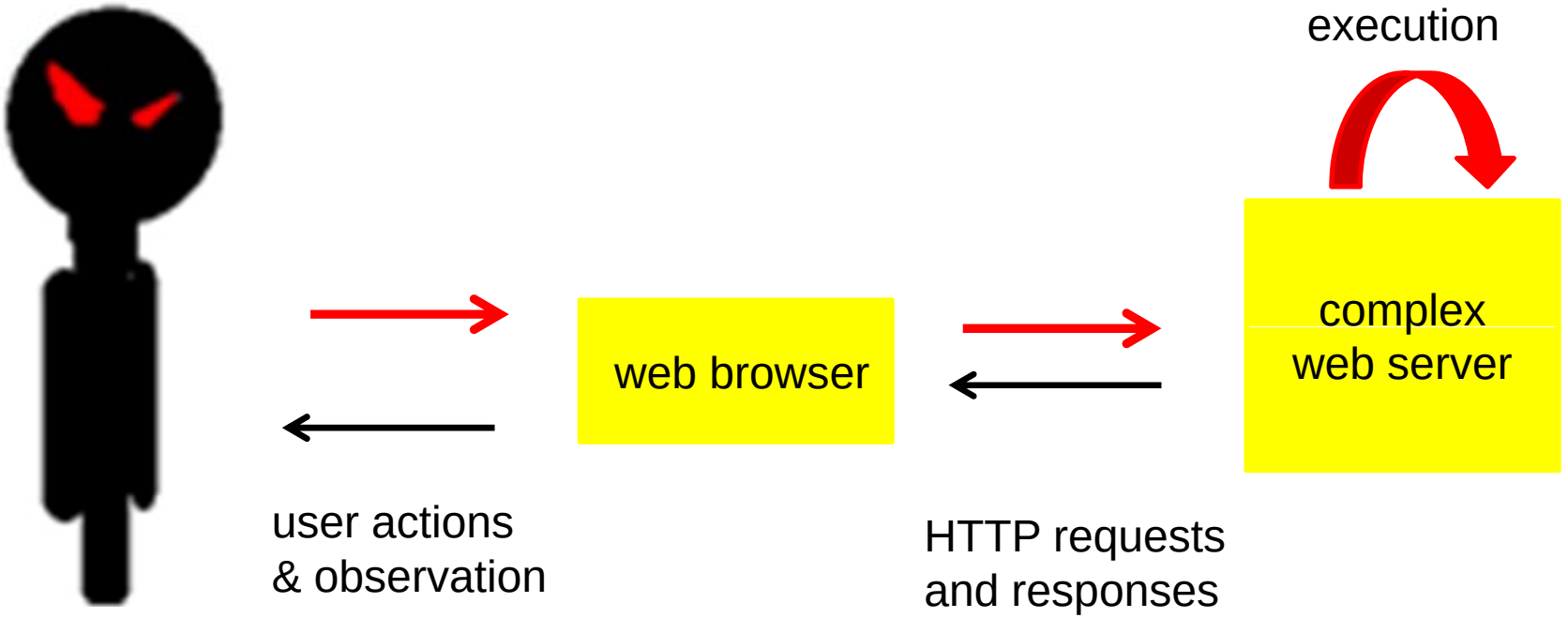
3. dynamically created web pages

Additional security worries?

- *execution in the server based on (possibly malicious) input from user*

- *more complexity in the server*
  - complexity is bad for security

dynamically created webpages



Web 2.0

Web 2.0 saw a big rise in user-generated content for the web

Eg

- web forums
- wikipedia and other wiki's

Additional security worries?

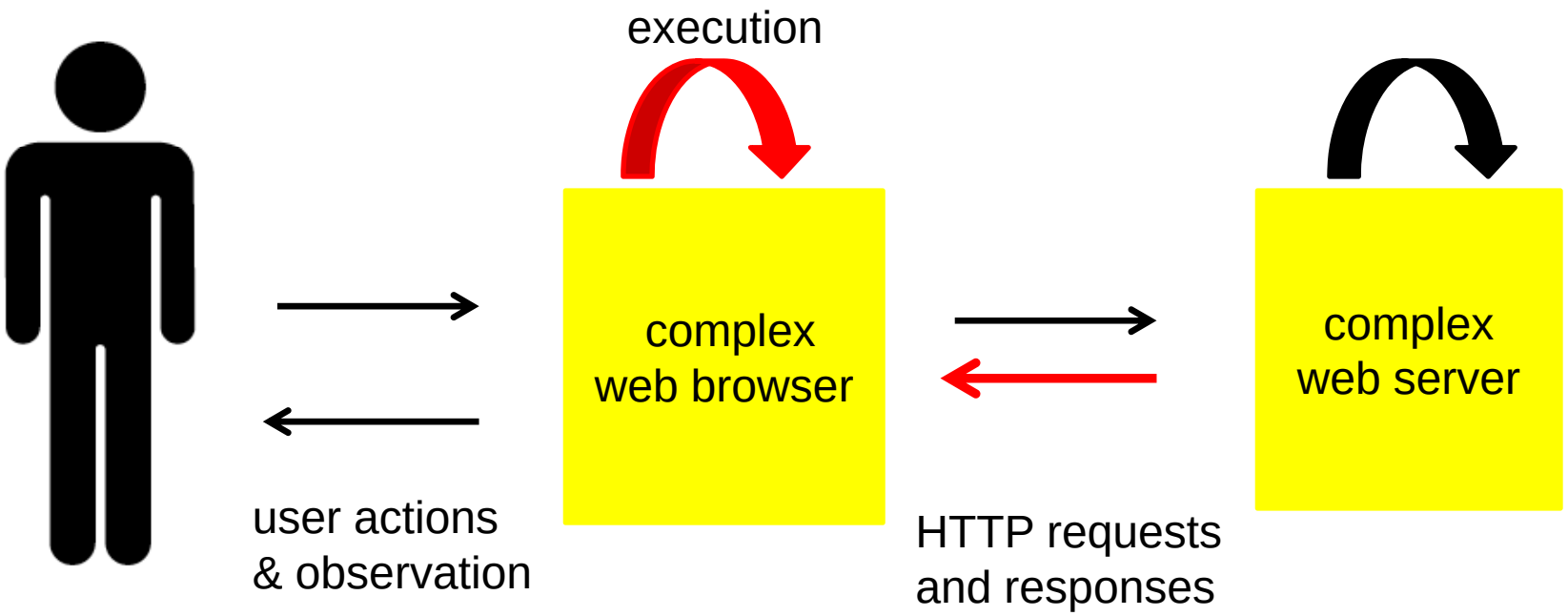
- *more opportunities for attacker to include malicious input*
  - *some of this input will be shown to other users*
    - so attacker can get other users to click on links
- *more authentication/integrity problems:*
  - where does the info you see on a website *really* come from?
- *more complexity in the server*

4. dynamic web pages

- ie executable contents *in* web pages
  - that is executed in the client's browser

  So web pages are not only generated dynamically by the server,
  but the web pages themselves are dynamic
- browser now also includes an execution engine
- typical languages used
  - JavaScript, Flash, Java, ActiveX

dynamic webpages



4. dynamic web pages

Additional security worries?

- *untrusted code executed in the user's browser*
- *server can't tell if the user or the executable content is responsible for HTTP requests*
- *more complexity, now at the client side*

Trend: more third party content

Web sites include ever increasing amounts of third party content, eg

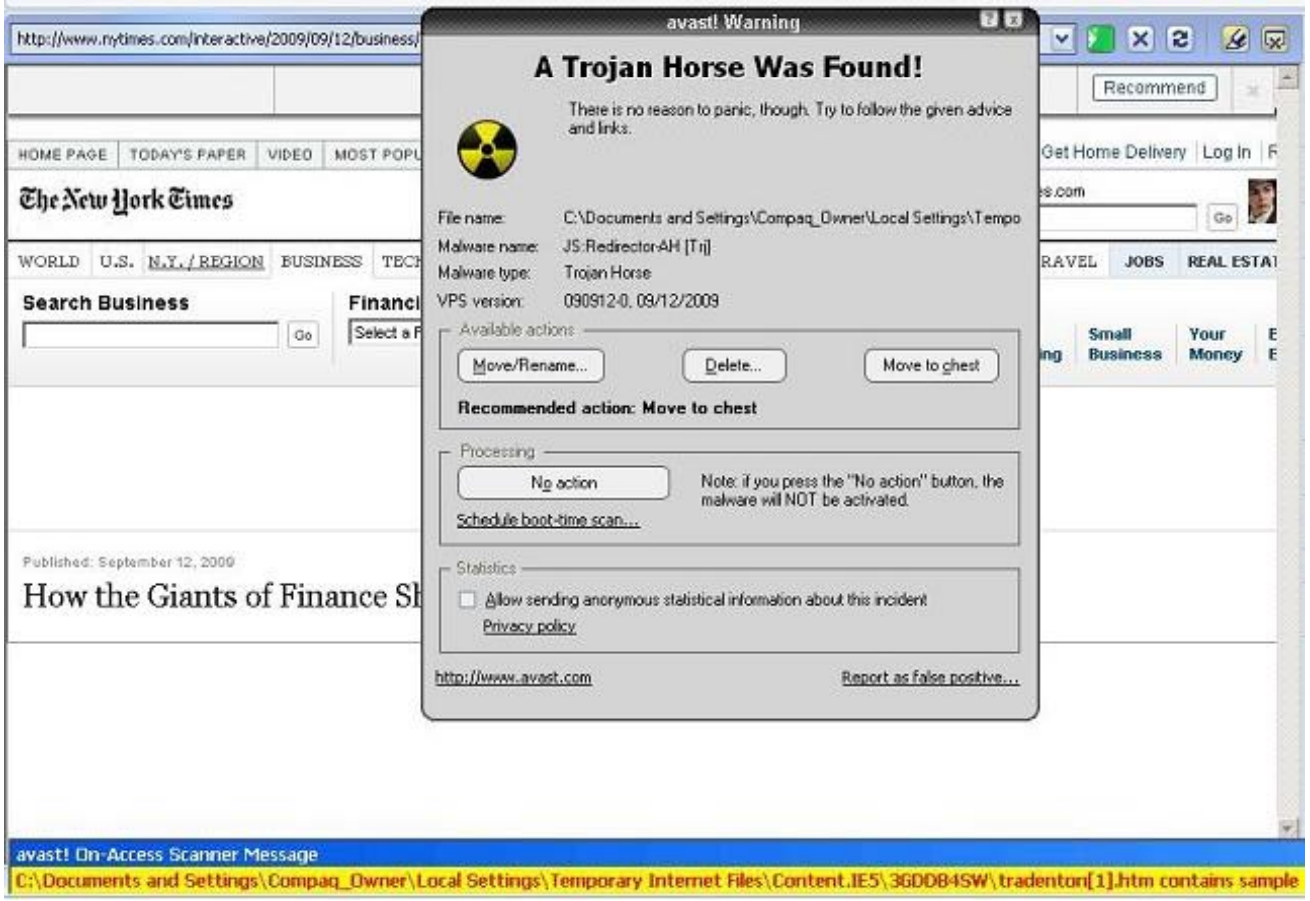
- google search bar or google map
- advertising from third parties
- tracing cookies from advertising networks
- web application mash-ups

Additional security threats?

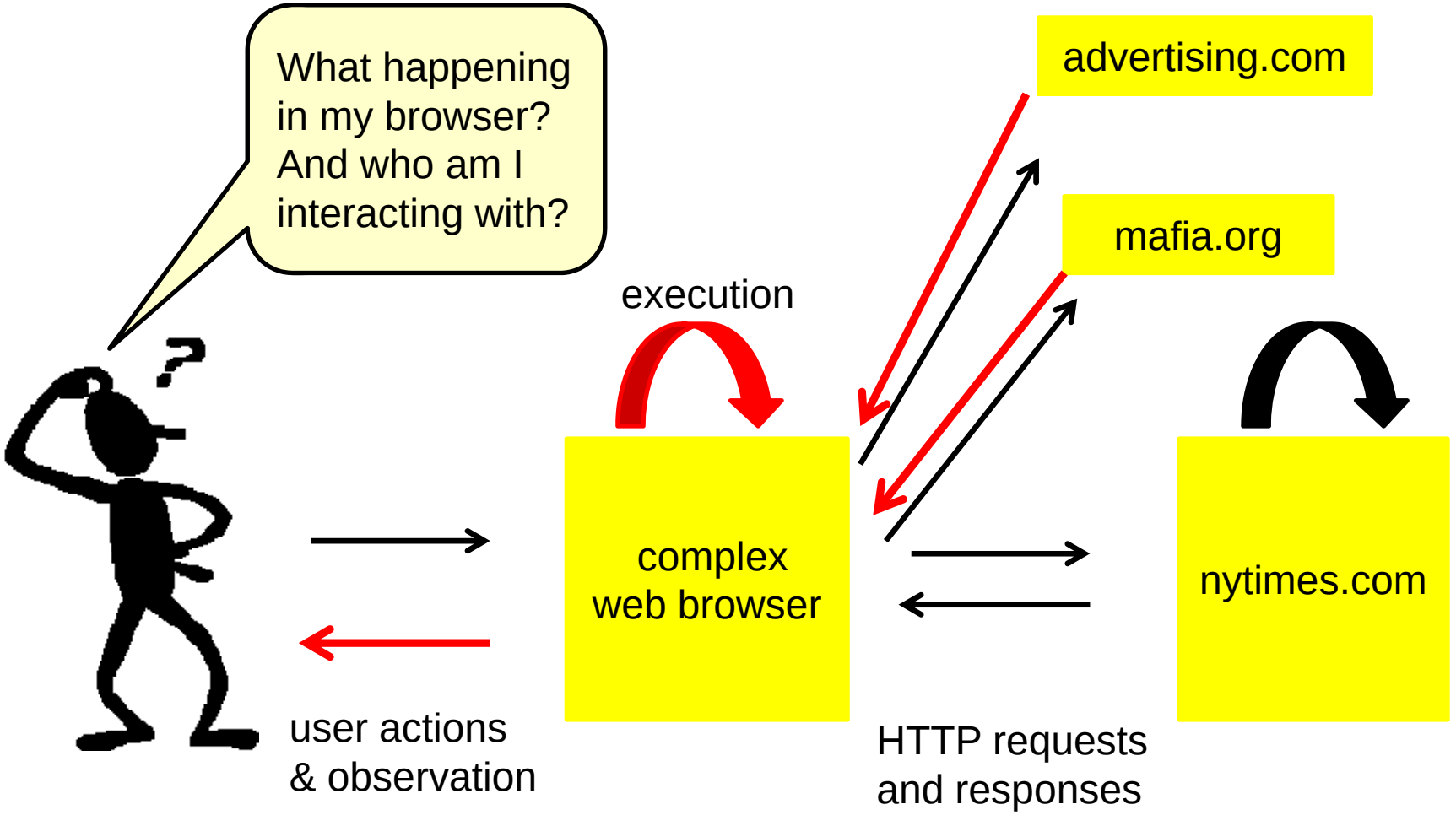
- *more risks to privacy*
- *more opportunities for attackers, via malicious 3rd party content*

Dangers of executable third-party content

JavaScript executing in a malicious advert included on New York Times web site creating a fake security pop-up



dynamic webpages and 3rd party content



the death of operating systems?

- Given the complexity - and power - of web applications,
  operating systems become less important,
  and might eventually become extinct?

- *All you need for everything is a browser, Gmail, and GoogleDocs?*

Still more dynamic: Ajax

Wilder client side execution with Ajax

(Asynchronous JavaScript and XML)

Code in browser runs independently of user actions (asynchronously)

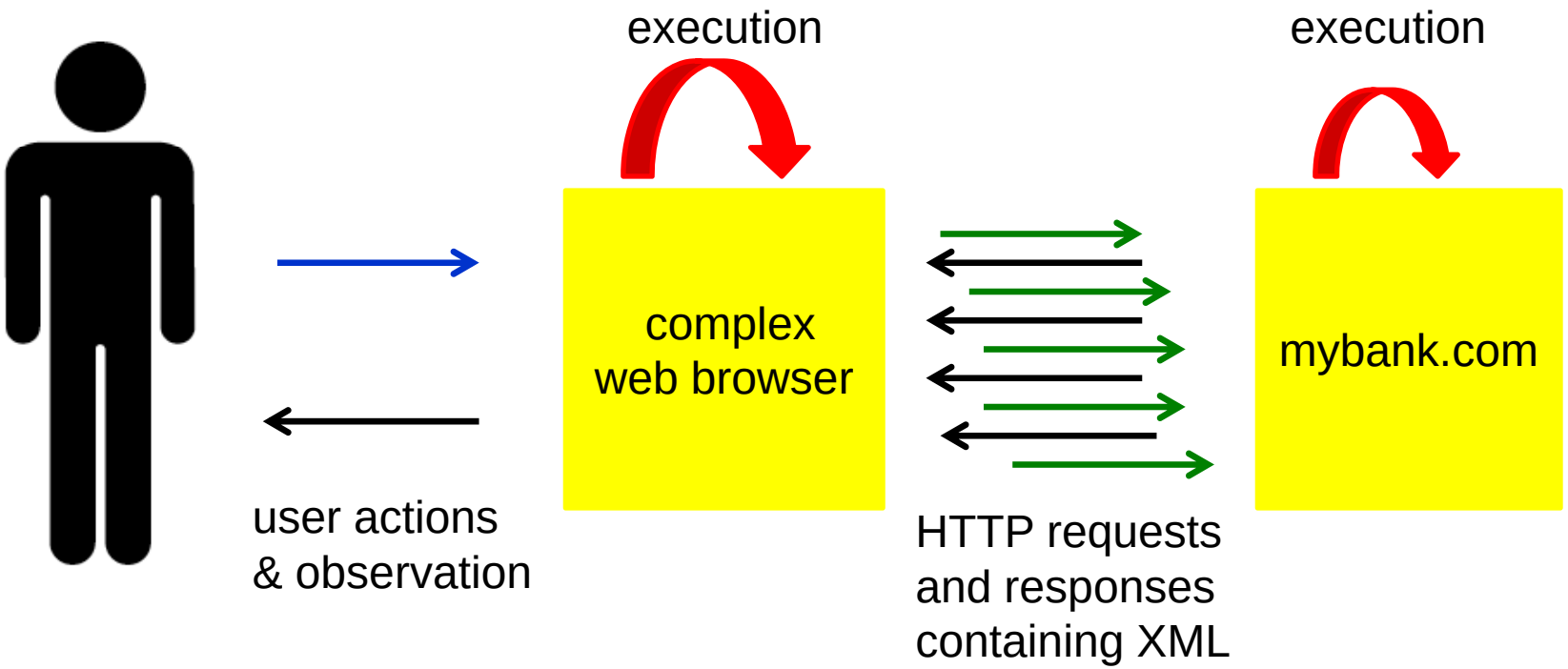
and uses XML for interacting with code in server

- classic example: word completion in Google search bar as you type
  - though this example is not really asynchronous

Confusingly, Ajax need not be asynchronous, and need not use XML.

Asynchronously interacting code executing in browser & server

Asynchronous: HTTP requests not necessarily caused by user actions



XML

XML (eXtensible Markup Language) allows custom extensions for encoding documents and other data in a format that is both human-readable and machine readable.

HTML and XML use very similar syntax for tags **`<tag>...</tag>`**

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,

eg **`<artist>Lady Gaga</artist>`**

Future: web 3.0 aka semantic web ?

Vision of future web where information is not just text, but is enriched with semantics (meaning) that can be interpreted and processed by machines, and not just by humans.

Note that some processing of information on the web is already automated,

eg by web crawlers that offer price comparisons

Still, richer information on web sites could make this easier and more powerful.

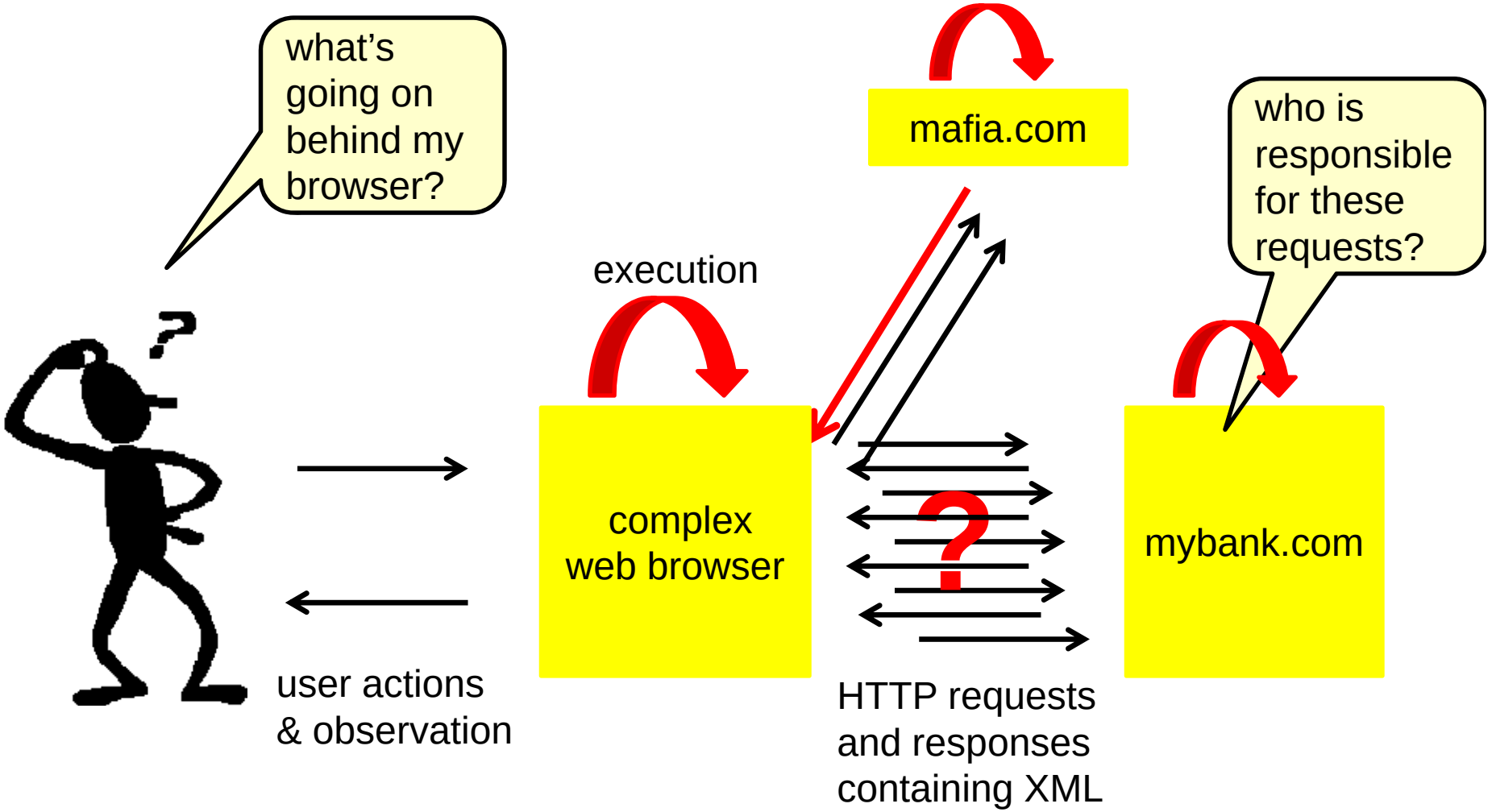
Still more dynamic: Ajax

Additional security worries?

- *more untrusted executable content*
- *more complexity*
- *user less in control*
- *even less clear who is responsible for HTTP requests:*

  the user or the asynchronously executing JavaScript?

Asynchronously interacting code executing in browser & server



overall trends

Two trends that seem unstoppable

- more complexity
- ever more dynamic content

Both are bad news for security...

protocols & formats

IP

IP (Internet Protocol) is the network-level 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 numeric IP addresses

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 971
[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

A client starts a HTTP dialogue by giving a URL to his browser

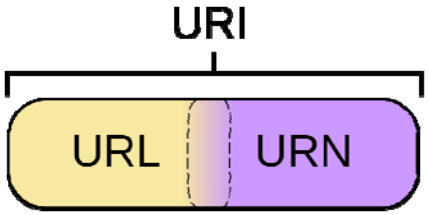
scheme://login:password@address:port/path/to/resource?query_string#fragment

1 2 3 4 5 6 7

1. scheme/protocol name, eg http, https, ftp, file, ...
2. credentials: username and password (optional)
3. address: IP address or DNS name
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, eg 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.

Strictly (pedantically) speaking, a URL is a special kind of URI.
There are 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.

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 a URL a user types, or link he clicks, into an HTTP request

HTTP requests

A request has the form

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

HTTP supports many methods, but only two are important

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

HTTP response

A request 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

HTML

The body of an HTTP response typically consists of HTML (Hypertext Markup Language)

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

- **`<href ..>`** to add clickable link
- **`<img ..>`** to include an image
- **`<script ..>`** to include link to a script that is automatically downloaded and executed

forms in HTML

Forms in HTML allow user to pass parameters (aka a query string)
in an HTTP request

```
<form method="GET" action= "http://ru.nl/register.php">
    Name: <input type="text" name="Your name">
    Email: <input type="text" name="Your email address">
     <input type="submit" value="Click here to submit">
</form>
```

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 idempotent comes from mathematics:

  eg. rounding a number is idempotent, squaring it is not .

GET vs POST

Query string (“parameters”) treated differently for GET and POST

- GET: query string is passed *in URL*

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

- POST: query string is 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
- should not be used for sensitive data!
- have a maximum length

POST requests

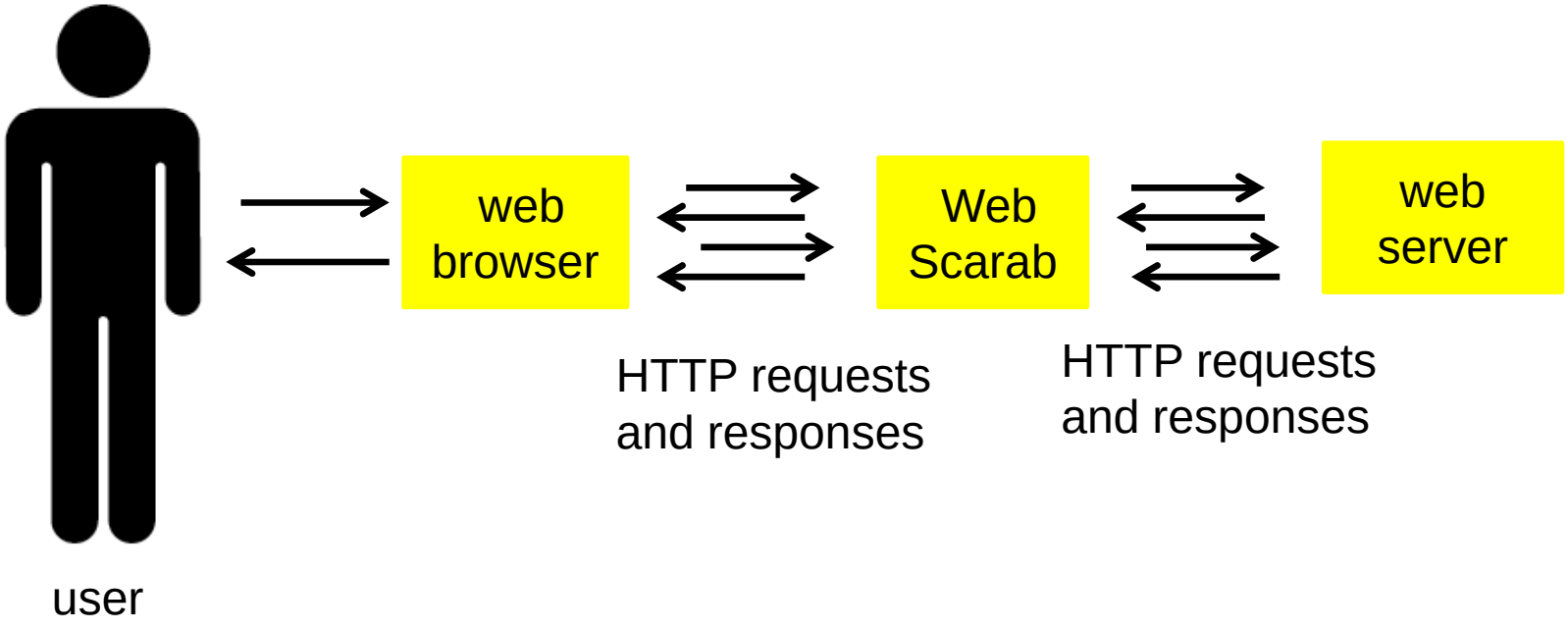
- are never cached
- cannot be bookmarked
- do not end up in browser history
- have no restrictions on length

Looking at HTTP traffic and 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.

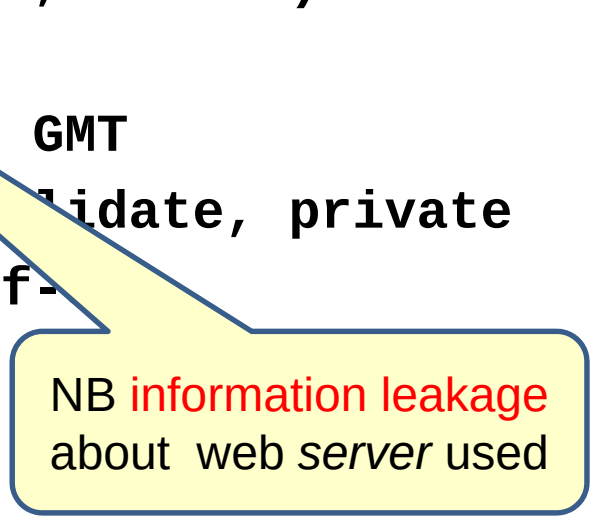
- You can inspect HTTP requests and responses using tools such as OWASP WebScarab, which we will use in the lab assignments

WebScarab as proxy



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-
Content-Length: 5687
Set-Cookie: keyword=value,...
<HTML>
....
</HTML>
```



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

For you to do

- Get the book
- Read chapters 1, 5.1, and 7.1.1
- Try to install WebGoat and WebScarab on your PC or laptop
  See links & info on the course webpage

Look at HTTP traffic generated by browsing at

`http://www.cs.ru.nl/~erikpoll/sws2/index.hmtl`

`http://www.cs.ru.nl/~erikpoll/sws2/demo`

`http://www.cs.ru.nl/~erikpoll/sws2/demo/demo.hmtl`

Observe that parameters end up in URL or body for GET and POST, and observe that some characters (such as @ in email address) are encoded