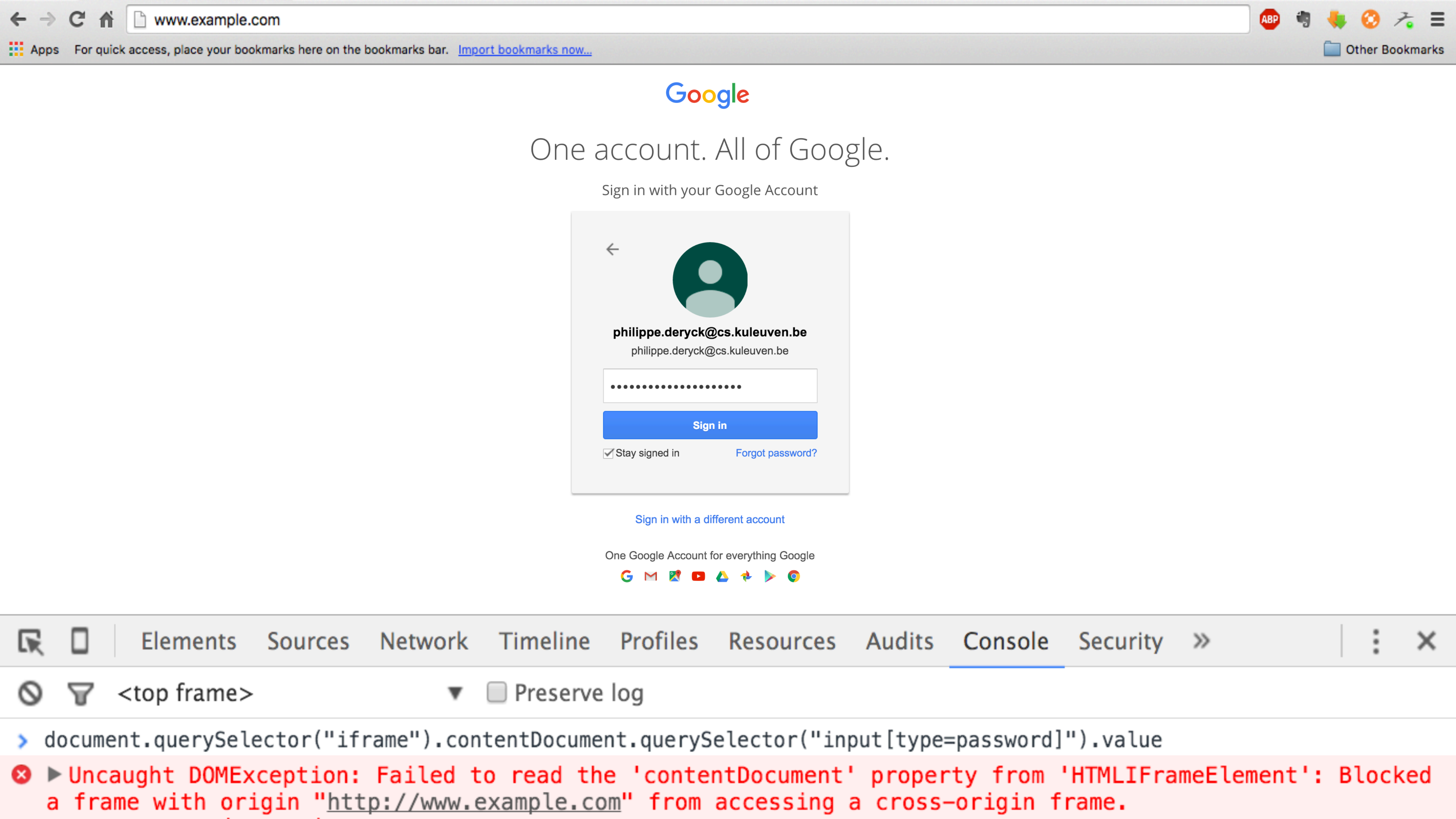
# The Web's Security Model

Philippe De Ryck

@PhilippeDeRyck https://www.websec.be



www.example.com
Apps
For quick access, place your bookmarks here on the bookmarks bar.
Import bookmarks now...
Other Bookmarks
Google
One account. All of Google.
Sign in with your Google Account
philippe.deryck@cs.kuleuven.be
philippe.deryck@cs.kuleuven.be
Sign in
Stay signed in
Forgot password?
Sign in with a different account
One Google Account for everything Google
Elements
Sources
Network
Timeline
Profiles
Resources
Audits
Console
Security
<top frame>
Preserve log

```
> document.querySelector("iframe").contentDocument.querySelector("input[type=password]").value
✖ ▶Uncaught DOMException: Failed to read the 'contentDocument' property from 'HTMLIFrameElement': Blocked
  a frame with origin "http://www.example.com" from accessing a cross-origin frame.
```

## The Agenda for Today

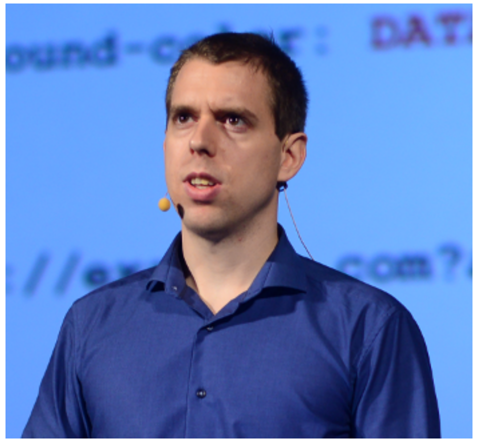
- The Same-Origin Policy
  - Setting a baseline with very relevant 20 year old technology
- Third-Party Content Integration
  - Frame and script-based integration
- Session Management
  - Cookies and the unavoidable CSRF attacks
- Accessing Cross-Origin APIs
  - Extending the SOP with server-driven policies
- Conclusion

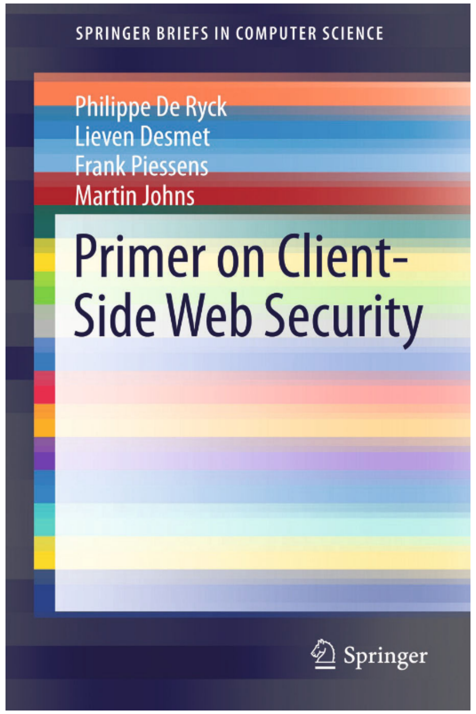
# About Me – Philippe De Ryck

- Postdoctoral Researcher @ DistriNet (KU Leuven)
  - PhD on client-side Web security
  - Expert in the broad field of Web security
  - Main author of the *Primer on Client-Side Web Security*

- Running the Web Security training program
  - Dissemination of knowledge and research results
  - Public training courses and targeted in-house training
  - Target audiences include industry and researchers

@PhilippeDeRyck https://www.websec.be




SPRINGER BRIEFS IN COMPUTER SCIENCE
Philippe De Ryck
Lieven Desmet
Frank Piessens
Martin Johns
Primer on Client-Side Web Security
Springer

## The Same-Origin Policy

## Same-Origin Policy

- Separation based on origin
  - Default security policy enforced by the browser
  - Restricts the interactions between contexts of different origins
  - Protects applications from unintended interactions
  - First appeared in browsers in 1995, and still going strong

**ORIGIN**

The triple <scheme, host, port> derived from the document's URL. For *http://example.org/forum/*, the origin is <*http, example.org, 80*>

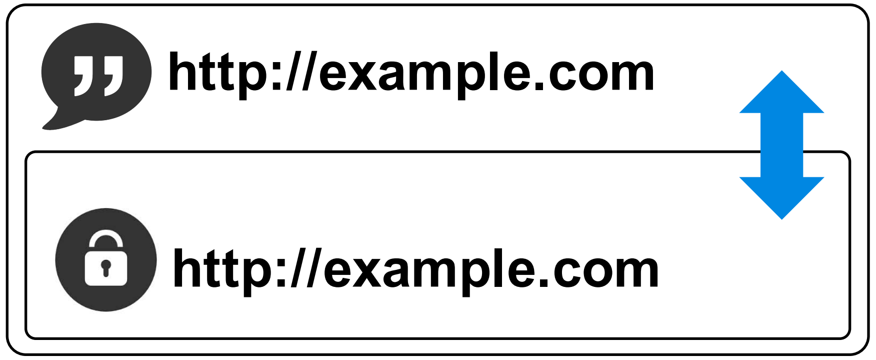
**SAME-ORIGIN POLICY**

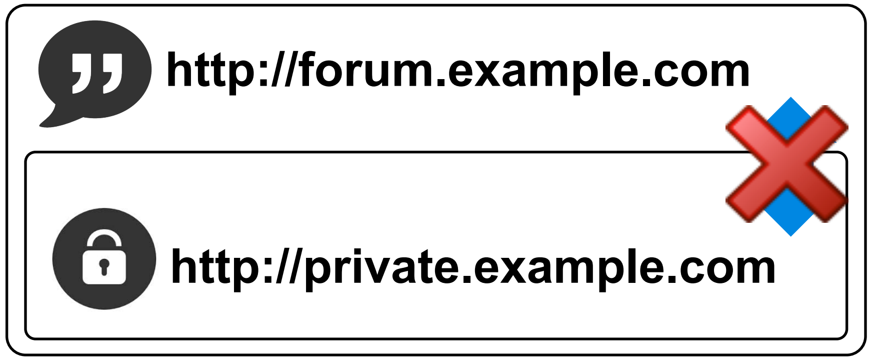
Content retrieved from one origin can freely interact with other content from that origin, but interactions with content from other origins are restricted

# Examples of the Same-Origin Policy

**SAME-ORIGIN POLICY**

Content retrieved from one origin can freely interact with other content from that origin, but interactions with content from other origins are restricted

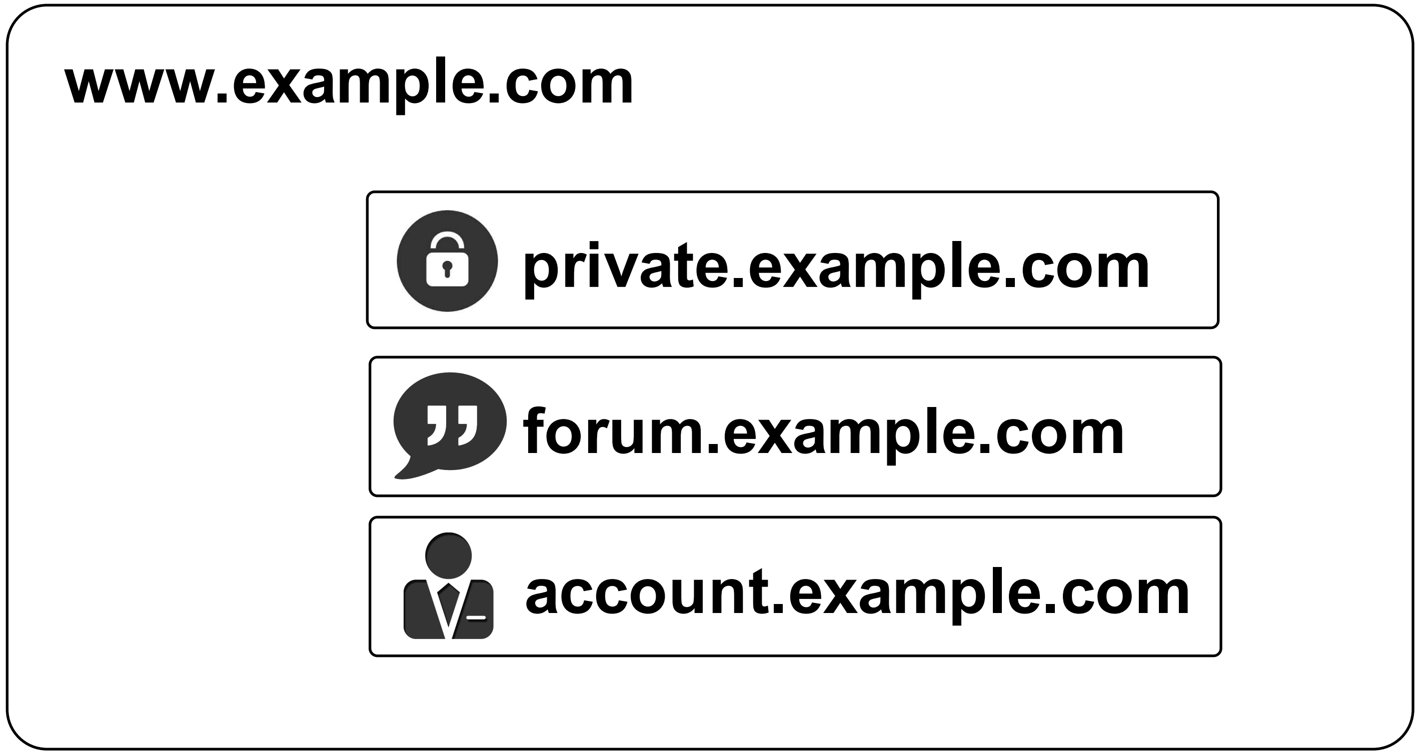




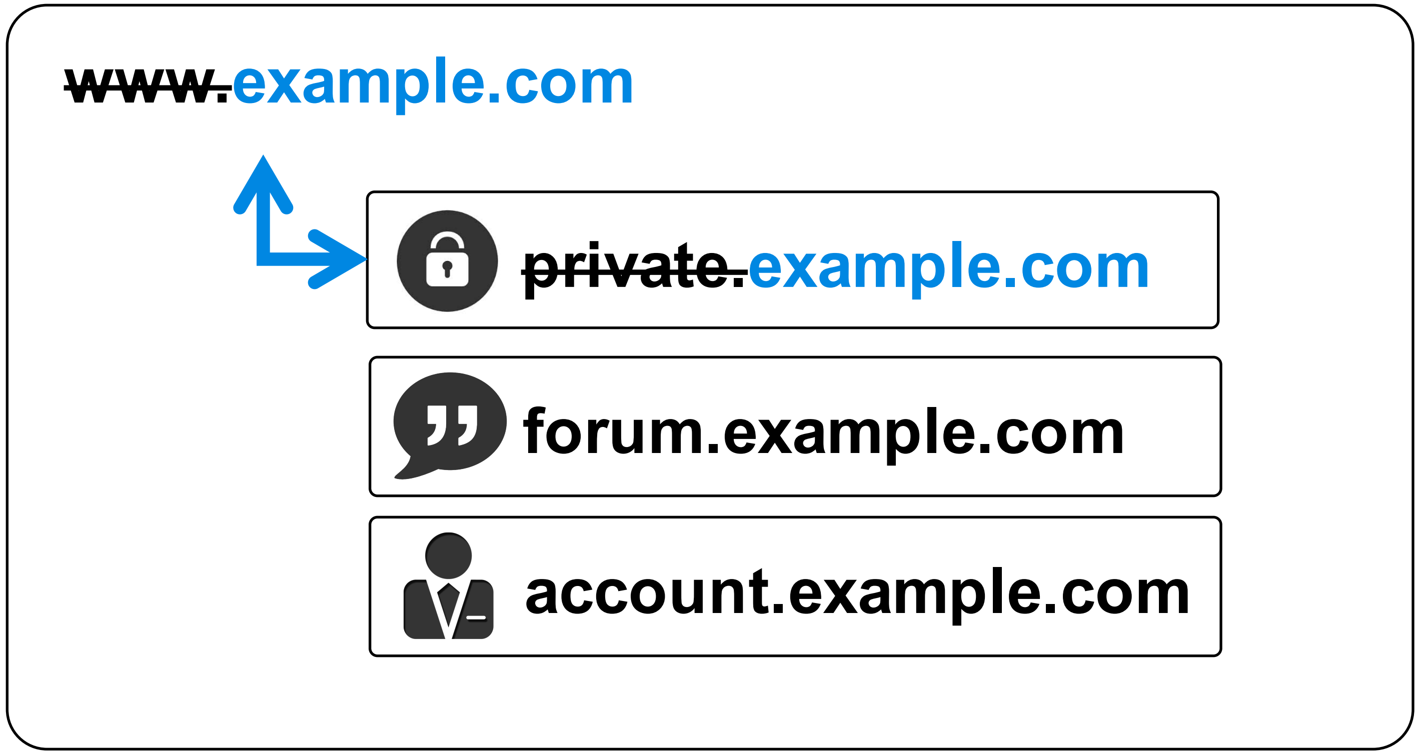
## Domains vs Subdomains

- Subdomains
  - E.g. ***private.example.com*** vs ***forum.example.com***
  - Considered different origin
  - Origin can be relaxed to ***example.com*** using *document.domain*
  - Possibility to use cookies on ***example.com***
- Completely separate domains
  - E.g. ***private.example.com*** vs ***exampleforum.com***
  - Considered different origin, without possibility of relaxation
  - No possibility of shared cookies

## Subdomains and Domain Relaxation



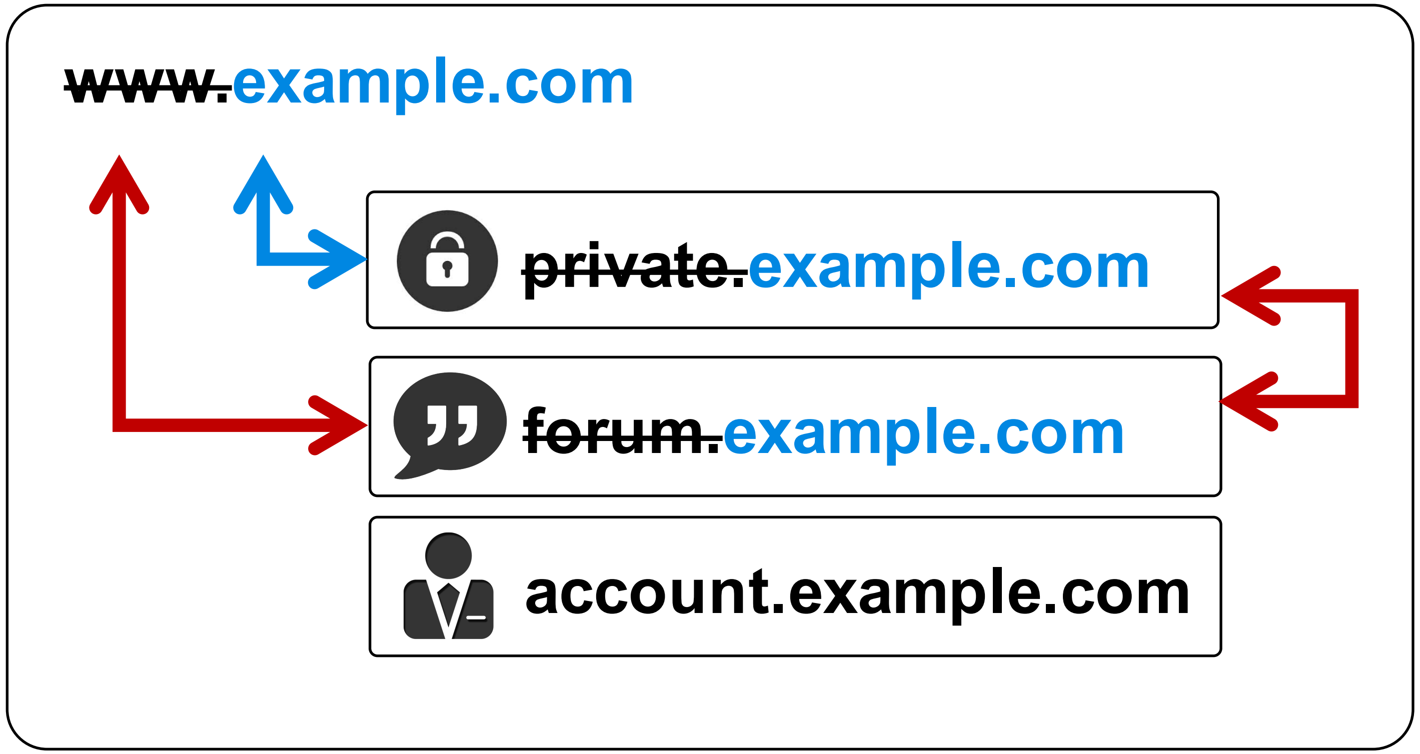
# Subdomains and Domain Relaxation



**DOMAIN RELAXATION**

```
document.domain = "example.com";
```

## Subdomains and Domain Relaxation



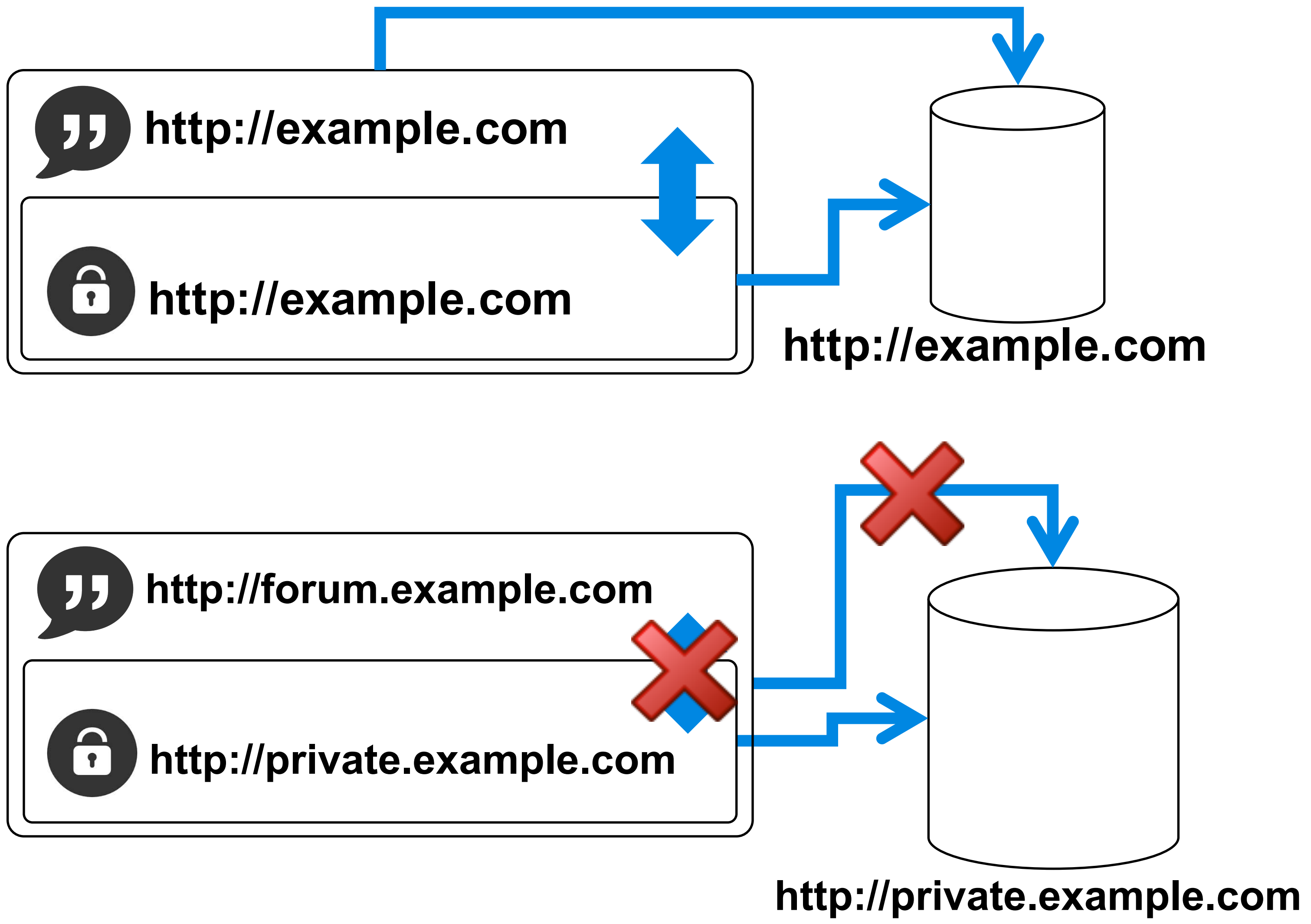
**Domain Relaxation**

```
document.domain = "example.com";
```

# But the SOP Is More than Context Isolation

**Same-Origin Policy**

Content retrieved from one origin can freely interact with other content from that origin, but interactions with content from other origins are restricted



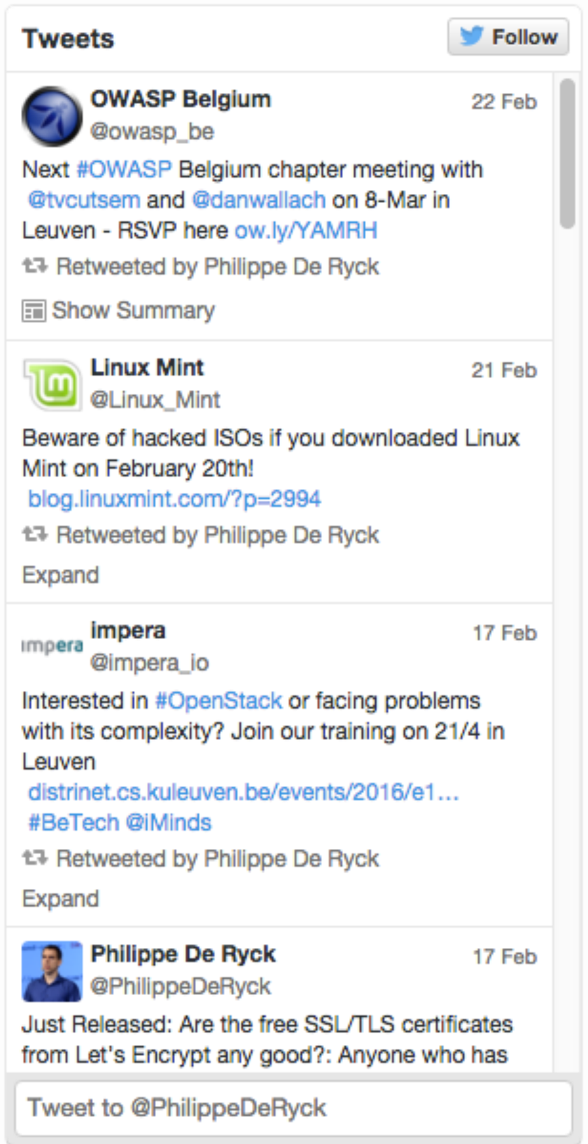
## Origin-Protected Resources

- Modern browsers offer plenty of origin-protected resources
  - The DOM and all its contents
  - Client-side storage facilities
    - Web storage, In-browser file systems, Indexed DB
  - Permissions to various ”invasive” features
    - Geolocation, full-screen capabilities, media capture, …
  - WebRTC video and audio streams
  - Ability to load and inspect resources from same-origin servers
  - Ability to send XHR requests without restrictions

- You want to be in control of what happens in your origin

## Third-Party Content Integration

# Third-Party Content Integration



Tweets  Follow

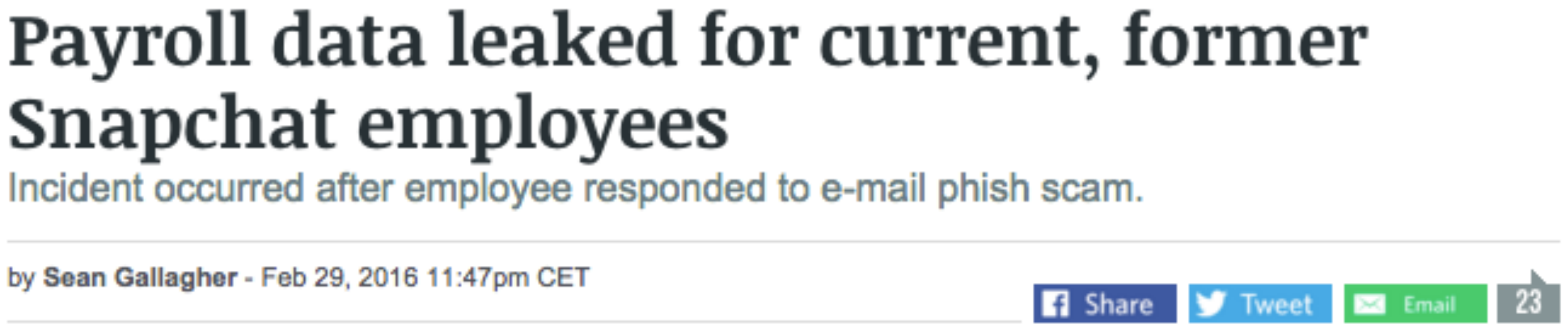
OWASP Belgium @owasp_be 22 Feb
Next #OWASP Belgium chapter meeting with @tvcutsem and @danwallach on 8-Mar in Leuven - RSVP here ow.ly/YAMRH
Retweeted by Philippe De Ryck
Show Summary

Linux Mint @Linux_Mint 21 Feb
Beware of hacked ISOs if you downloaded Linux Mint on February 20th! blog.linuxmint.com/?p=2994
Retweeted by Philippe De Ryck
Expand

impera @impera_io 17 Feb
Interested in #OpenStack or facing problems with its complexity? Join our training on 21/4 in Leuven distrinet.cs.kuleuven.be/events/2016/e1… #BeTech @iMinds
Retweeted by Philippe De Ryck
Expand

Philippe De Ryck @PhilippeDeRyck 17 Feb
Just Released: Are the free SSL/TLS certificates from Let's Encrypt any good?: Anyone who has

Tweet to @PhilippeDeRyck

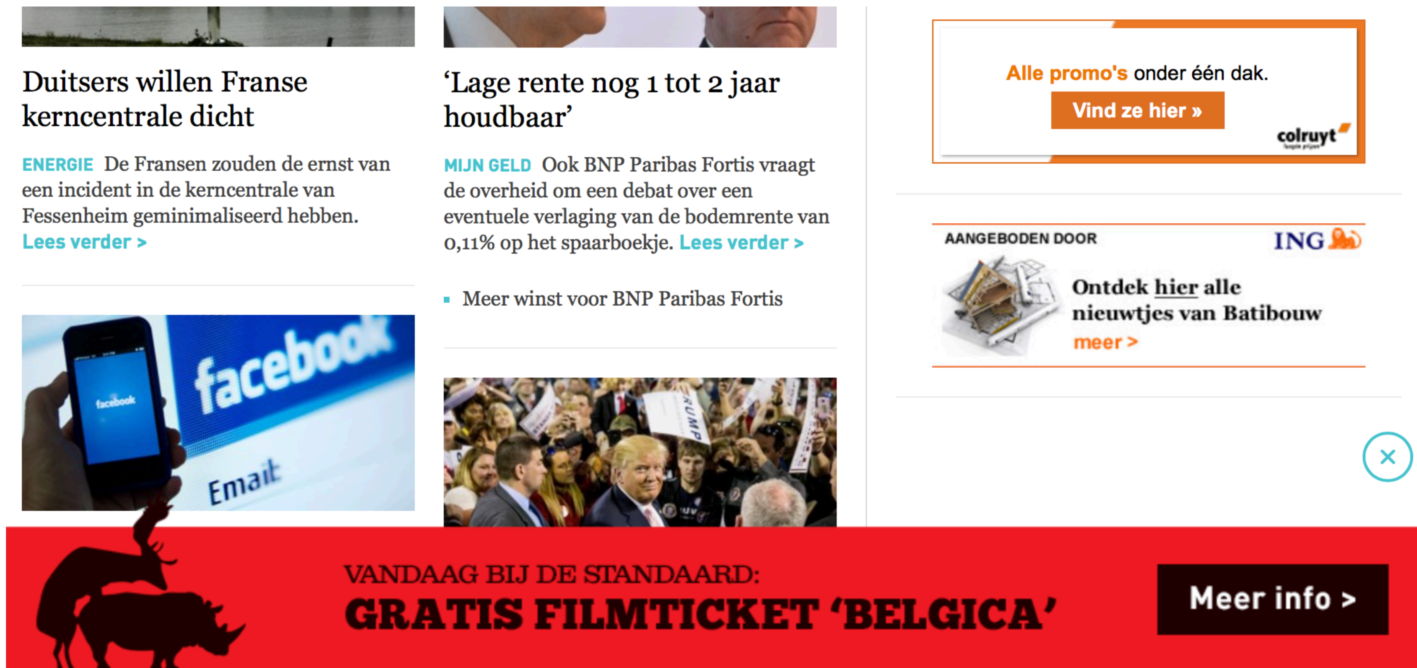


Payroll data leaked for current, former Snapchat employees

Incident occurred after employee responded to e-mail phish scam.

by Sean Gallagher - Feb 29, 2016 11:47pm CET

Share  Tweet  Email  23



Duitsers willen Franse kerncentrale dicht

ENERGIE De Fransen zouden de ernst van een incident in de kerncentrale van Fessenheim geminimaliseerd hebben. Lees verder >

‘Lage rente nog 1 tot 2 jaar houdbaar’

MIJN GELD Ook BNP Paribas Fortis vraagt de overheid om een debat over een eventuele verlaging van de bodemrente van 0,11% op het spaarboekje. Lees verder >

- Meer winst voor BNP Paribas Fortis

Alle promo's onder één dak.
Vind ze hier »
colruyt

AANGEBODEN DOOR ING
Ontdek hier alle nieuwtjes van Batibouw
meer >

VANDAAG BIJ DE STANDAARD:
GRATIS FILMTICKET ‘BELGICA’
Meer info >

## Integration of Third-Party Code

- Two mechanisms to integrate code
  - Embedding an *iframe*, which hosts a separate document
  - Directly including JavaScript code using the <*script*> tag

- Iframes
  - Each iframe is a different context, with a separate origin
  - Preserves the security boundaries, but may hinder interaction
- Scripts
  - Scripts are loaded and executed within the page's context
  - Violates the security boundaries of a document

## Iframe-based Content Integration

- Iframes are controlled by the same-origin policy
  - Documents with different origins are isolated by the SOP
  - Well-suited to integrate separate components (e.g. advertisements)
  - Allows you to apply the principle of least privilege
  - More difficult to achieve dynamic interaction

- HTML5 introduces the sandbox attribute
  - Supports disabling scripts, plugins, forms, etc.
  - Allows you to assign a unique origin to your content
  - Integrate untrusted content with a minimal set of capabilities

## Interaction between Contexts

- Related contexts
  - Documents can open popup windows, embed frames, etc.
  - Related cross-origin contexts are isolated by default
  - Limited interactions possible (navigation, messaging APIs, …)
- Navigation
  - Navigate child frame to different resource
  - Navigate parent frame, reloading the entire document
- Exposed APIs
  - Prime example: Web Messaging API, to support interaction

## Web Messaging API

- Messaging mechanism between contexts
  - Used for iframes, Web Workers, etc.
  - Event listener for receiving messages (opt-in mechanism)
  - API function for sending data (text, objects, etc.)

- Security considerations
  - Specify origin of receiver to prevent leaking of content
  - Check origin of sender to prevent malicious use
  - Validate incoming content before using data to prevent injection attacks

# Web Messaging API

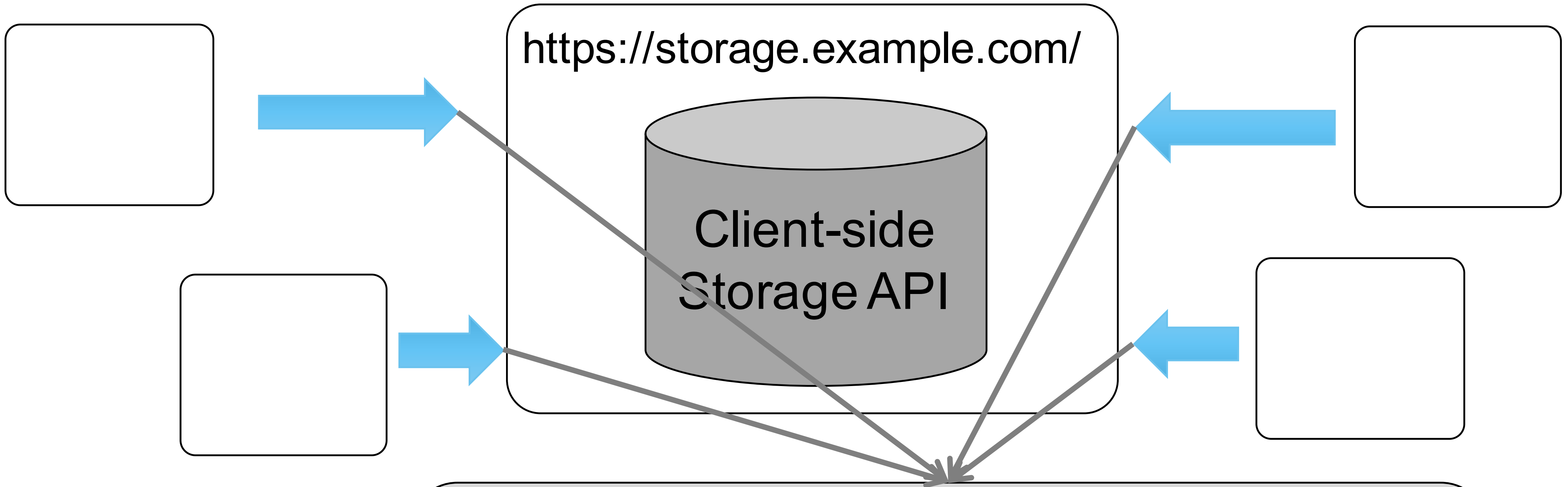
## Sending Messages

```
myframe.postMessage(data,'http://test.example.com');
```

## Receiving Messages

```
var handler = function(event) {
  if(event.origin ==
    'http://www.example.com') {
      alert(event.data);
  }
}
window.addEventListener('message', handler, false);
```

## Example: a Client-side Storage Facility



**Accessing local storage through Web Messaging allows enforcing access control and content inspection**

## Script-based Content Integration

- No security boundaries offered by browser
  - Scripts are executed in the context that loads them
  - No boundaries between remote and local scripts
  - Full access to the client-side context, including local resources

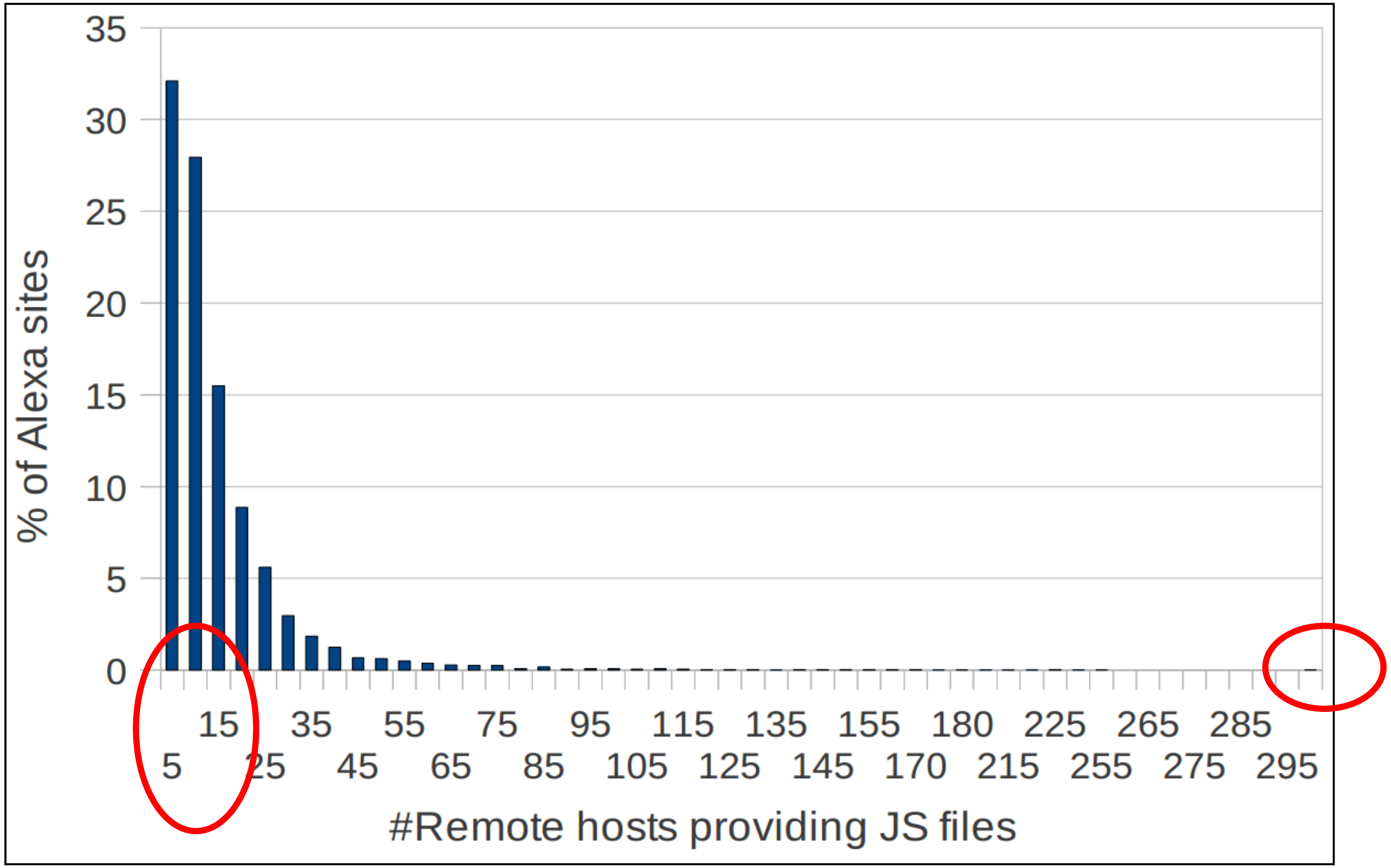
- Potentially dangerous setup
  - No more control if you include scripts from all over the place
  - Which has unfortunately become common practice

## Large-scale Study of Remote JS Inclusions

"88.45% of the Alexa top 10,000 web sites included at least one remote JavaScript library"

*https://seclab.cs.ucsb.edu/media/uploads/papers/jsinclusions.pdf*

# Large-scale Study of Remote JS Inclusions



https://seclab.cs.ucsb.edu/media/uploads/papers/jsinclusions.pdf

## Safely Including Third-Party Code

- Leverage origin-based separation using iframes
  - Load the third-party script in a document with a different origin
  - SOP enforces isolation from the main origin and sensitive resources

- Example case at Dropbox
  - They use a chat widget from a third-party provider
  - Inclusion in the main *dropbox.com* origin is an unacceptable risk
  - Widget loaded in an iframe with origin *dbxsnapengage.com*
  - Communication happens with Web Messaging

*https://blogs.dropbox.com/tech/2015/09/csp-third-party-integrations-and-privilege-separation/*

## Reclaiming Control over Your Context

- Say hello to Content Security Policy (CSP)
  - **Main goal**: prevent XSS attacks from causing harm
  - Allows you to specify where remote content can be loaded from
  - Allows you to specify where outgoing requests can go to

- Policy specified by the server, enforced by the browser
  - Gives you control over your own code
  - Allows you to selectively load third-party code
    - And constrain what that code can do

## Alternative Approaches to Constrain Scripts

- Hosting scripts in your own origin
  - Difficult to deal with a highly dynamic codebase
- Safe JavaScript subsets
  - Requires compatibility with existing scripts
- Server-side rewriting
  - Requires control over the scripts to do the rewriting
- Browser-based sandboxing
  - Requires browser modifications, which is a deployment nightmare
- JavaScript-based sandboxing
  - Active research topic, may become possible in the coming years

## Session Management

## The Basics of Cookies

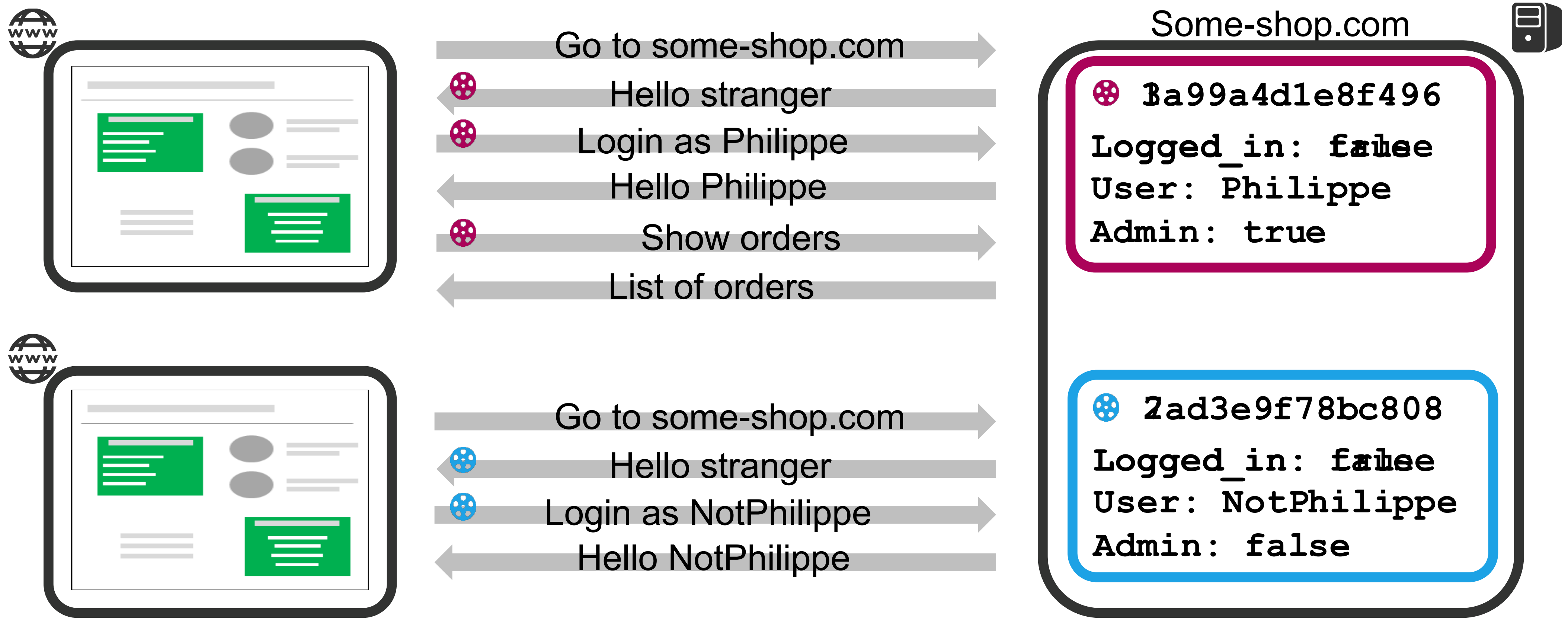
- An HTTP state management mechanism
  - Set by the server to the client through the Set-Cookie header
  - Offered by the client to the server through the Cookie header

```
Set-Cookie: name=value; Expires=Wed, 09 Jun 2021 10:18:14 GMT;
Domain=example.com; Secure; HttpOnly
```

```
Cookie: name=value
```

- Cookie properties
  - If no expiration date is set, it is removed when the browser closes
  - If no domain is set, it is only valid for the domain that issued it
    - Otherwise, it is sent to the current domain and all subdomains

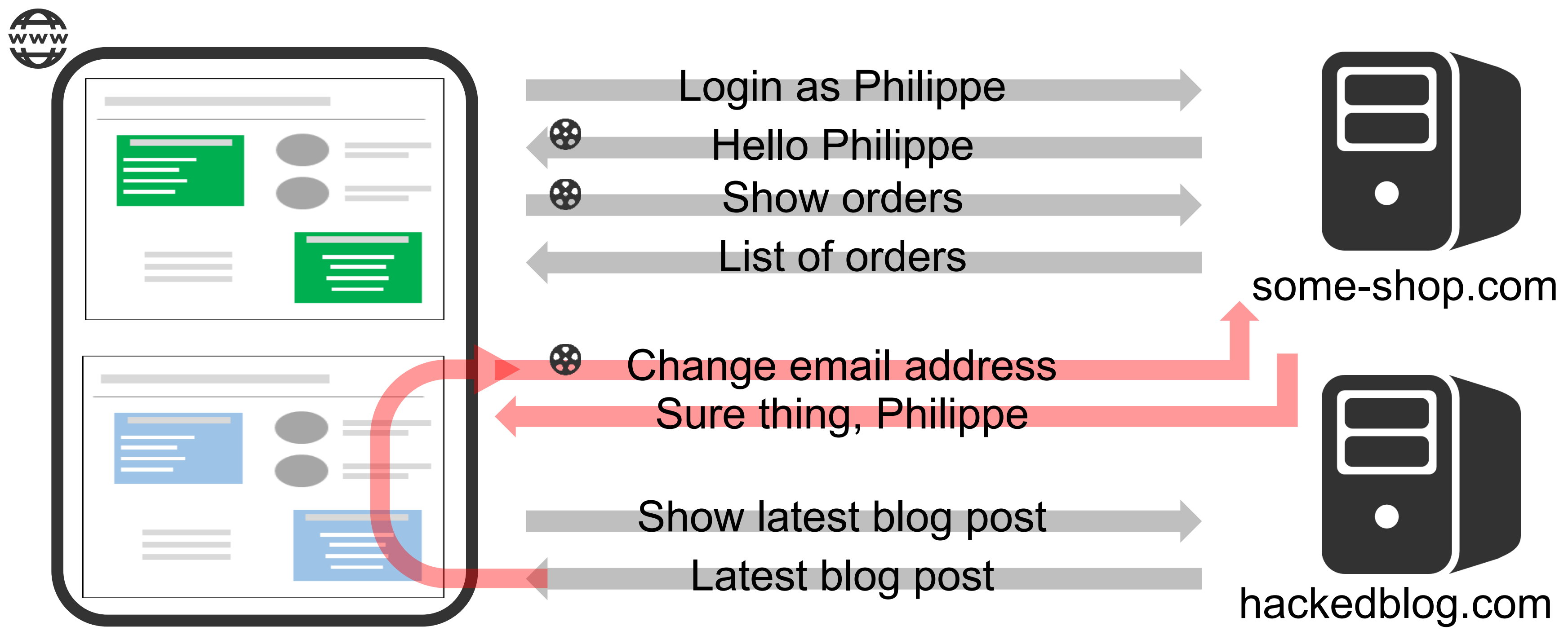
# Using Cookies to Manage Sessions



## Properties of Cookie-Based Sessions

- Session identifiers and objects are bearer tokens
  - The token represents ownership of the session
- Cookies are managed by the browser
  - Stored automatically
  - Automatically attached to every request, if the domain matches
- Common threats against cookie-based session management
  - Brute forcing a session identifier
  - Session hijacking and session fixation
  - Cross-Site Request Forgery

# Cross-Site Request Forgery Illustrated



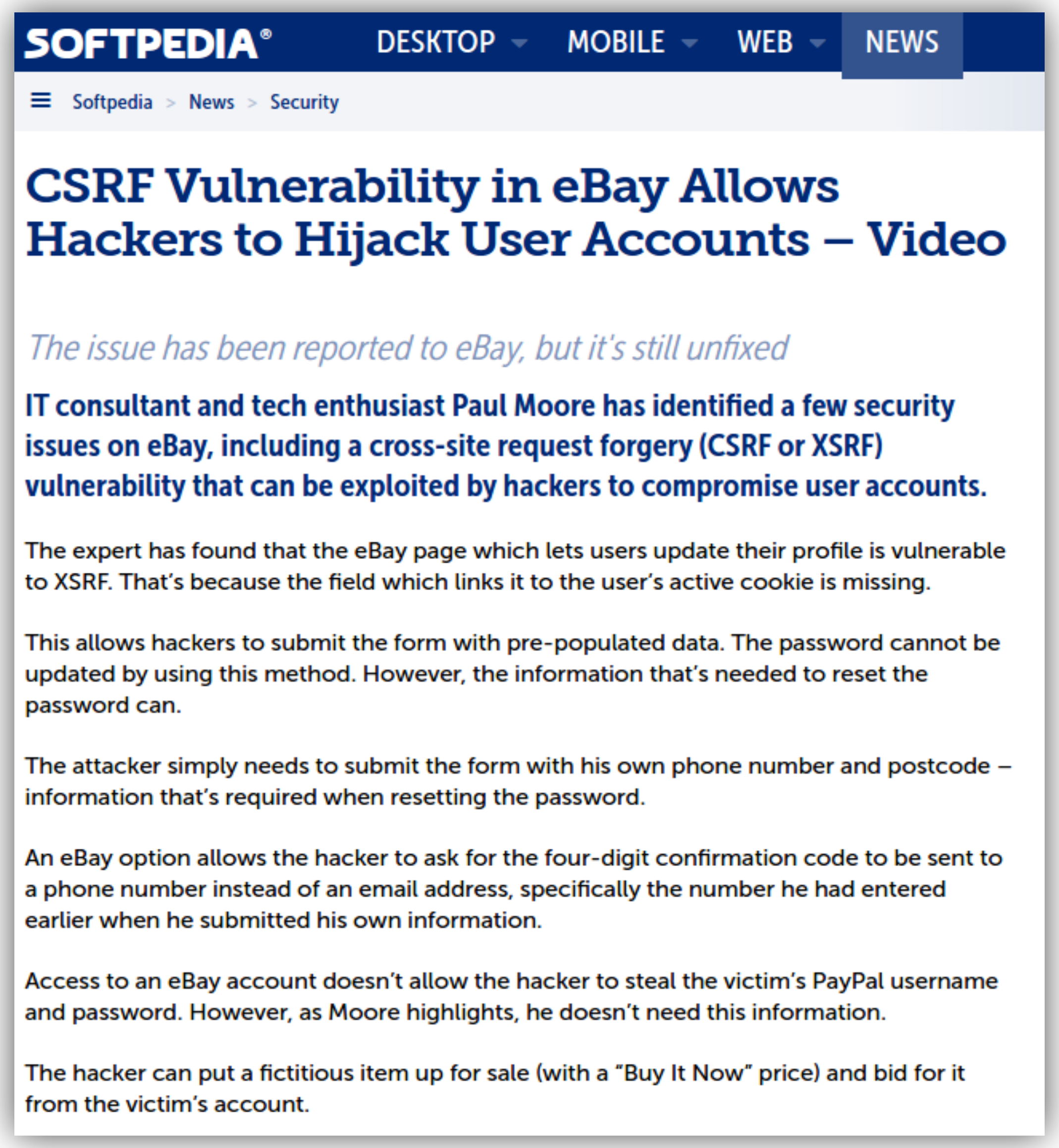
## The Essence of CSRF

- The server is confused about the intentions of the user
  - Malicious sites can trigger unintended requests from the browser
  - Consequence of the ambient authority carried by the cookie

- Common vulnerability
  - Illustrated by cases at Google, Facebook, eBay, …
  - Ranked #8 on OWASP top 10 (2013)

- Countermeasures require explicit action by the developer
  - Often only focus on POST / PUT / DELETE

# CSRF Examples



SOFTPEDIA® DESKTOP MOBILE WEB NEWS

Softpedia > News > Security

## CSRF Vulnerability in eBay Allows Hackers to Hijack User Accounts – Video

*The issue has been reported to eBay, but it's still unfixed*

**IT consultant and tech enthusiast Paul Moore has identified a few security issues on eBay, including a cross-site request forgery (CSRF or XSRF) vulnerability that can be exploited by hackers to compromise user accounts.**

The expert has found that the eBay page which lets users update their profile is vulnerable to XSRF. That's because the field which links it to the user's active cookie is missing.

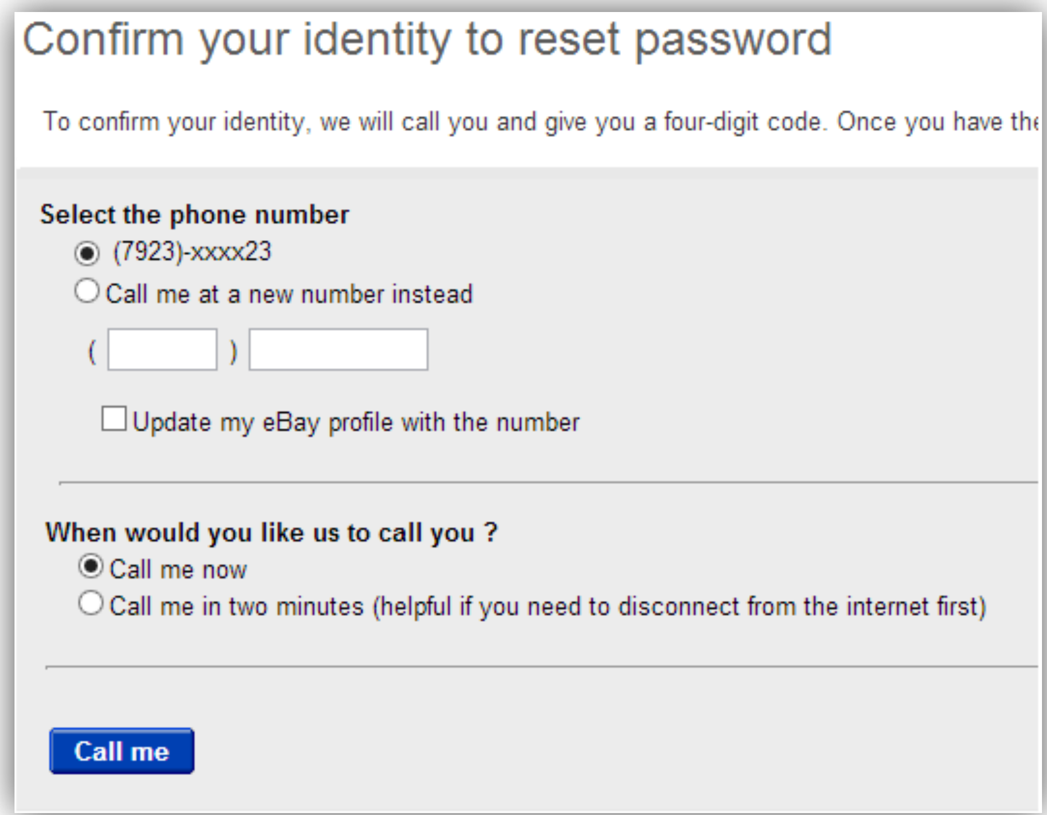
This allows hackers to submit the form with pre-populated data. The password cannot be updated by using this method. However, the information that's needed to reset the password can.

The attacker simply needs to submit the form with his own phone number and postcode – information that's required when resetting the password.

An eBay option allows the hacker to ask for the four-digit confirmation code to be sent to a phone number instead of an email address, specifically the number he had entered earlier when he submitted his own information.

Access to an eBay account doesn't allow the hacker to steal the victim's PayPal username and password. However, as Moore highlights, he doesn't need this information.

The hacker can put a fictitious item up for sale (with a "Buy It Now" price) and bid for it from the victim's account.



Confirm your identity to reset password

To confirm your identity, we will call you and give you a four-digit code. Once you have the

**Select the phone number**

[✓ (7923)-xxxx23]

Call me at a new number instead

( ) )

☐ Update my eBay profile with the number

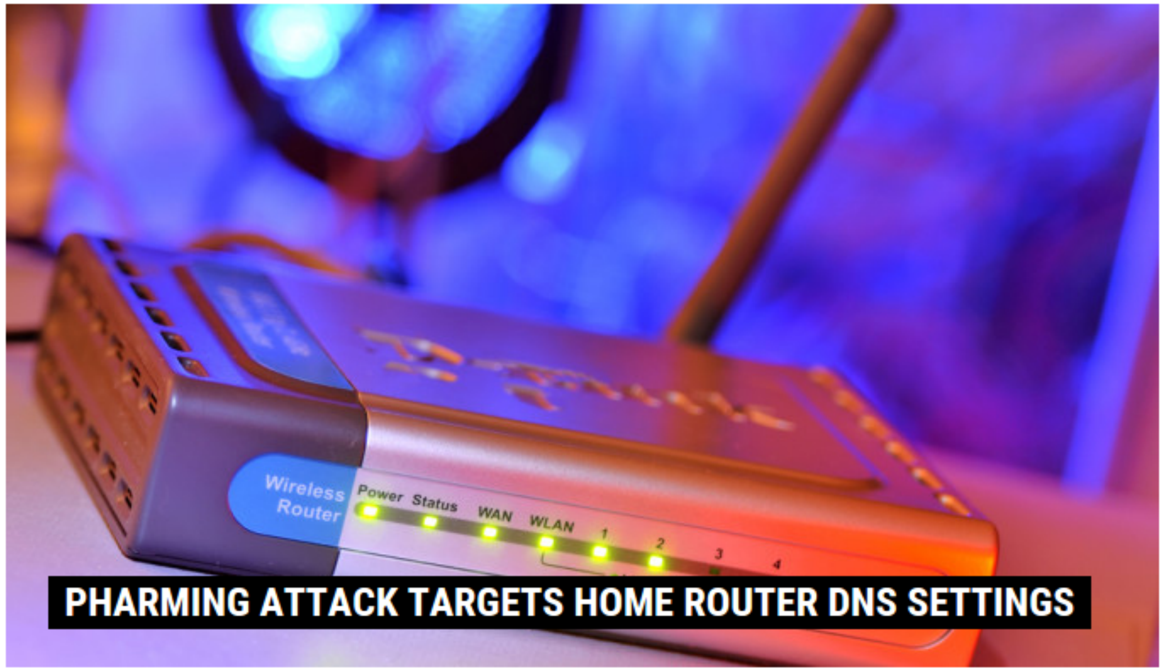
**When would you like us to call you ?**

[✓ Call me now]

Call me in two minutes (helpful if you need to disconnect from the internet first)

*http://news.softpedia.com/news/CSRF-Vulnerability-in-eBay-Allows-Hackers-to-Hijack-User-Accounts-Video-383316.shtml*

## CSRF Examples



PHARMING ATTACK TARGETS HOME ROUTER DNS SETTINGS

by Michael Mimoso  Follow @mike_mimoso  February 27, 2015 , 2:07 pm

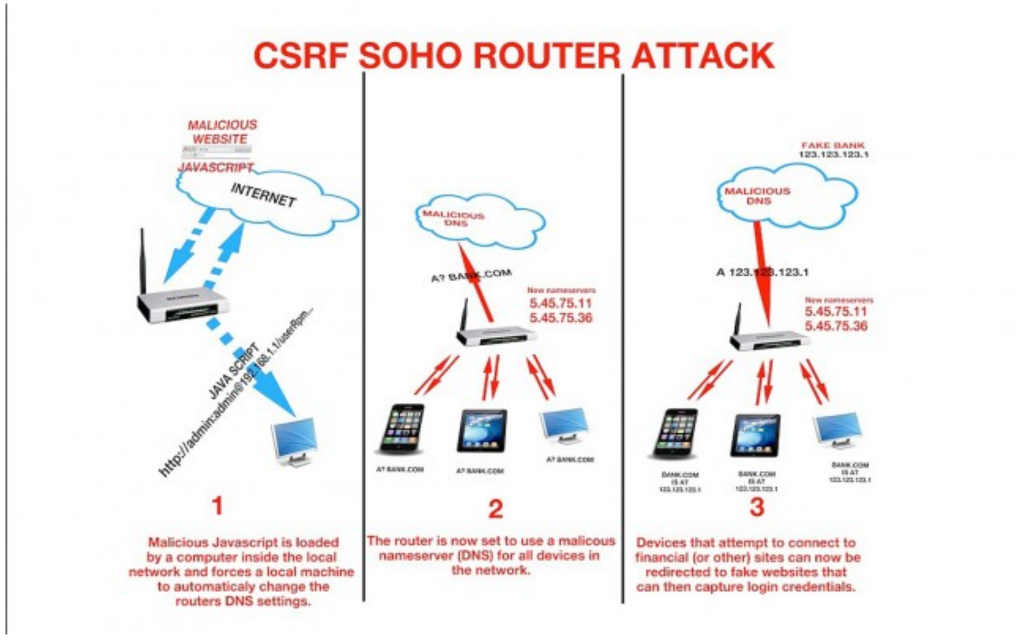
Pharming attacks are generally network-based intrusions where the ultimate goal is to redirect a victim's web traffic to a hacker-controlled webserver, generally through a malicious modification of DNS settings.

Some of these attacks, however, are starting to move to the web and have their beginnings with a spam or phishing email.

**Hackers hijack 300,000-plus wireless routers, make malicious changes**

Devices made by D-Link, Micronet, Tenda, and TP-Link hijacked in ongoing attack.

by **Dan Goodin** - Mar 3, 2014 8:42pm CET  Share  Tweet  116


CSRF SOHO ROUTER ATTACK
MALICIOUS WEBSITE
JAVASCRIPT
INTERNET
JAVA SCRIPT
http://admin:admin@192.168.1.1/userRpm...
1
Malicious Javascript is loaded by a computer inside the local network and forces a local machine to automaticaly change the routers DNS settings.
MALICIOUS DNS
A? BANK.COM
New nameservers
5.45.75.11
5.45.75.36
A? BANK.COM
A? BANK.COM
A? BANK.COM
2
The router is now set to use a malicous nameserver (DNS) for all devices in the network.
FAKE BANK 123.123.123.1
MALICIOUS DNS
A 123.123.123.1
New nameservers
5.45.75.11
5.45.75.36
BANK.COM IS AT 123.123.123.1
BANK.COM IS AT 123.123.123.1
BANK.COM IS AT 123.123.123.1
3
Devices that attempt to connect to financial (or other) sites can now be redirected to fake websites that can then capture login credentials.

Enlarge / Three phases of an attack that changes a router's DNS settings by exploiting a cross-site request vulnerability in the device's Web interface.

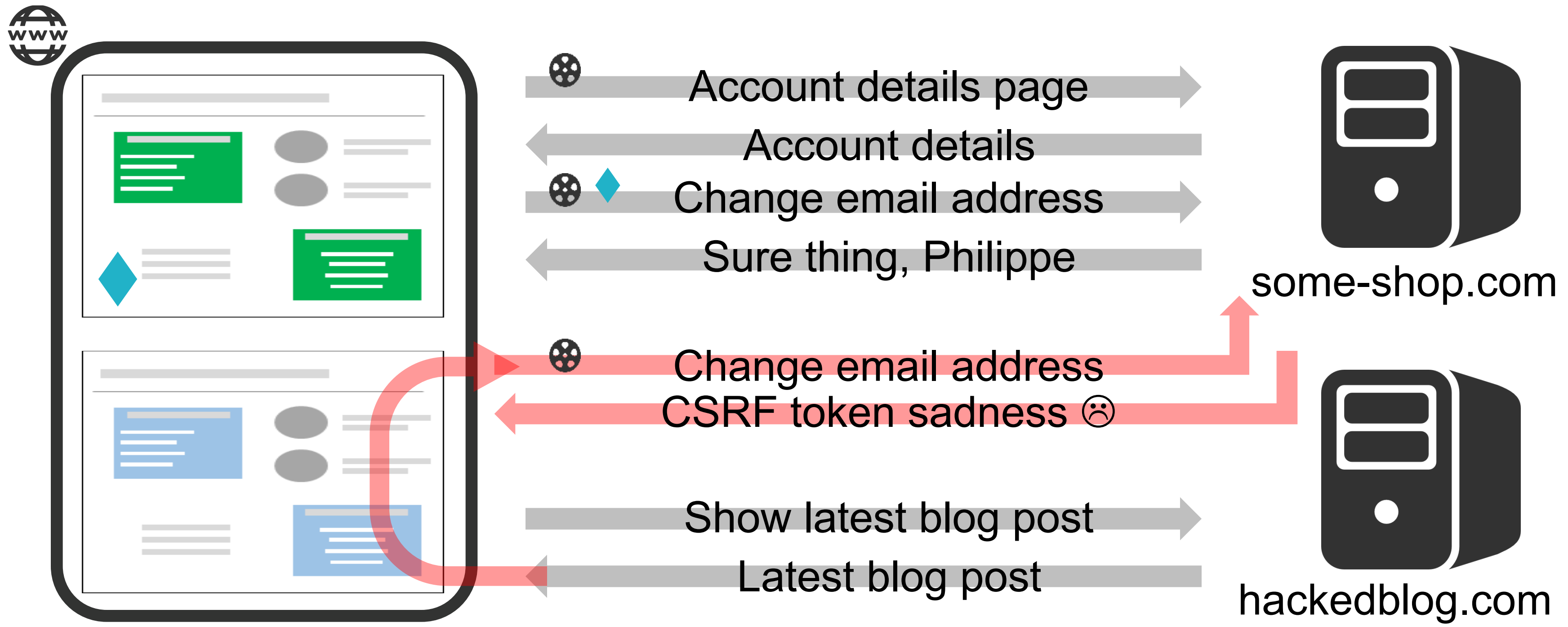
Team Cymru

Researchers said they have uncovered yet another mass compromise of home and small-office wireless routers, this one being used to make malicious configuration changes to more than 300,000 devices made by D-Link, Micronet, Tenda, TP-Link, and others.

*https://threatpost.com/pharming-attack-targets-home-router-dns-settings/111326*
*http://arstechnica.com/security/2014/03/hackers-hijack-300000-plus-wireless-routers-make-malicious-changes/*

## CSRF Defense 1: HTML tokens

- Hide token within the page, and check upon form submission
  - Same-Origin Policy keeps this token out of reach for the attacker



# CSRF Defense 1: HTML tokens

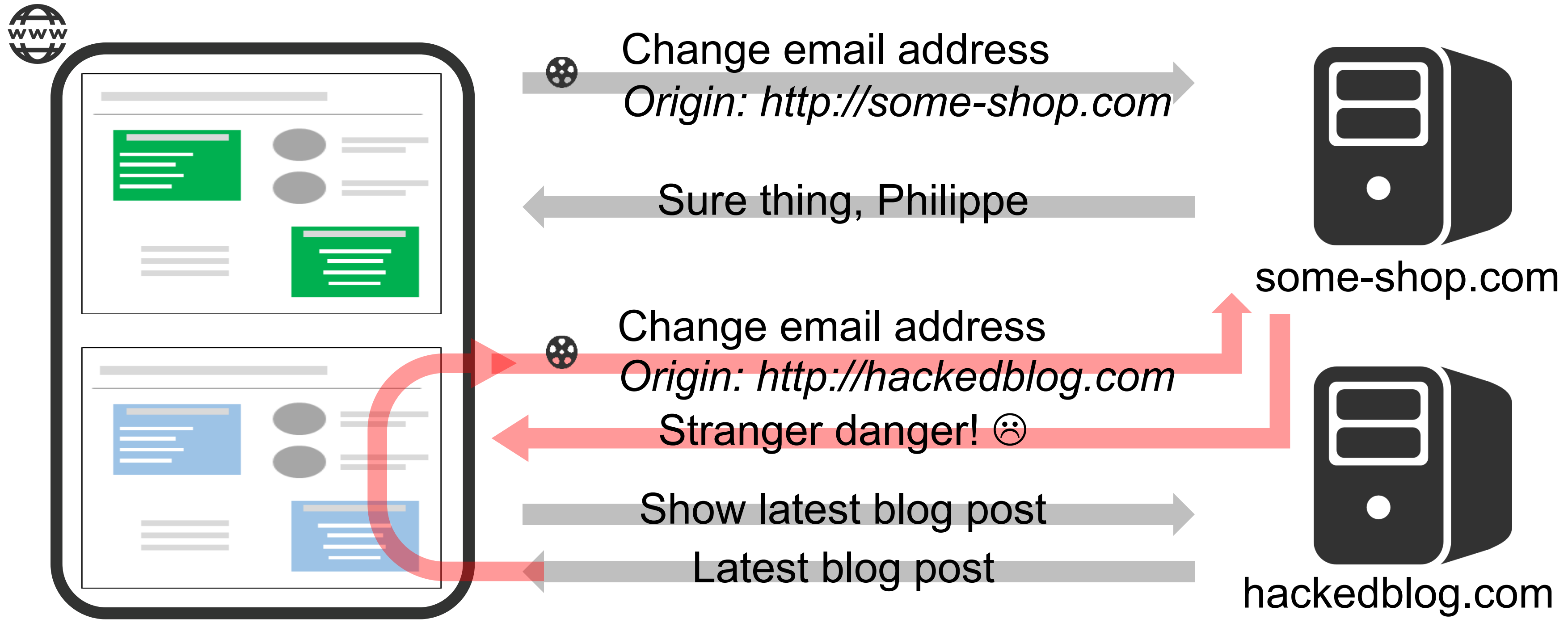
- Hide token within the page, and check upon form submission
  - Same-Origin Policy keeps this token out of reach for the attacker

**TOKEN-BASED APPROACH**

```
<form action="submit.php">
  <input type="hidden" name="token"
    value="qasfj8j12adsjadu2223" />
  …
</form>
```

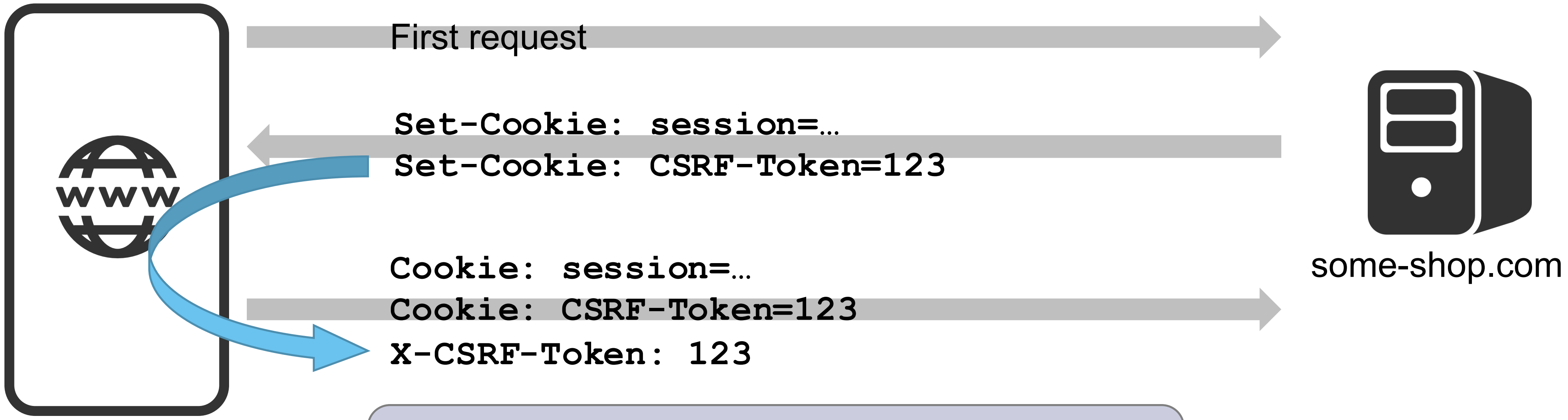
## CSRF Defense 2: Origin Header

- Check the origin header sent by the browser
  - Automatically added to state-changing requests (POST, PUT, DELETE)



## CSRF Defense 3: Transparent Tokens

- Transparent token stored in cookie, checked in header
  - Security depends on the ability to read the cookie from JavaScript



```
Set-Cookie: session=…
Set-Cookie: CSRF-Token=123
```

```
Cookie: session=…
Cookie: CSRF-Token=123
X-CSRF-Token: 123
```

**Only the JS code on the page can copy cookie value into header**

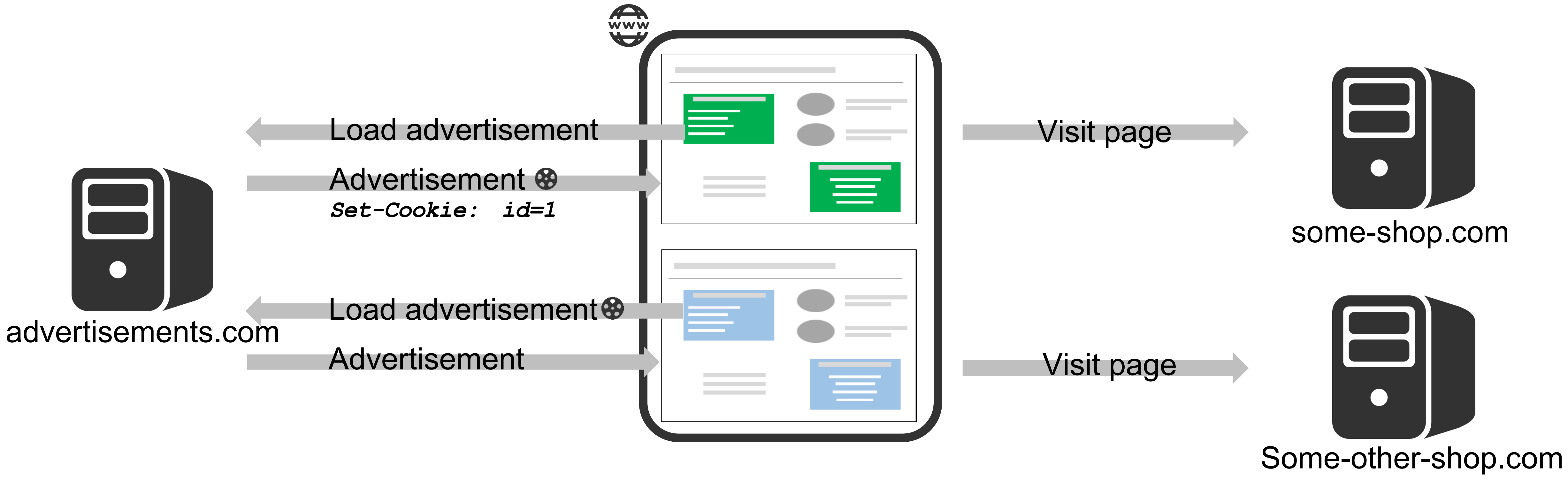
## Traditional Cookie-Based Session Management

- Still very common in Web applications
  - Well-supported by browsers and frameworks
  - Often deployed in an insecure fashion
  - Keep best practices in mind when using this mechanism

- Cookie-based session management best practices
  - Deploy your application over HTTPS
  - Set the HttpOnly and Secure flags
  - Host user-provided content on a different domain
  - Deploy CSRF defenses

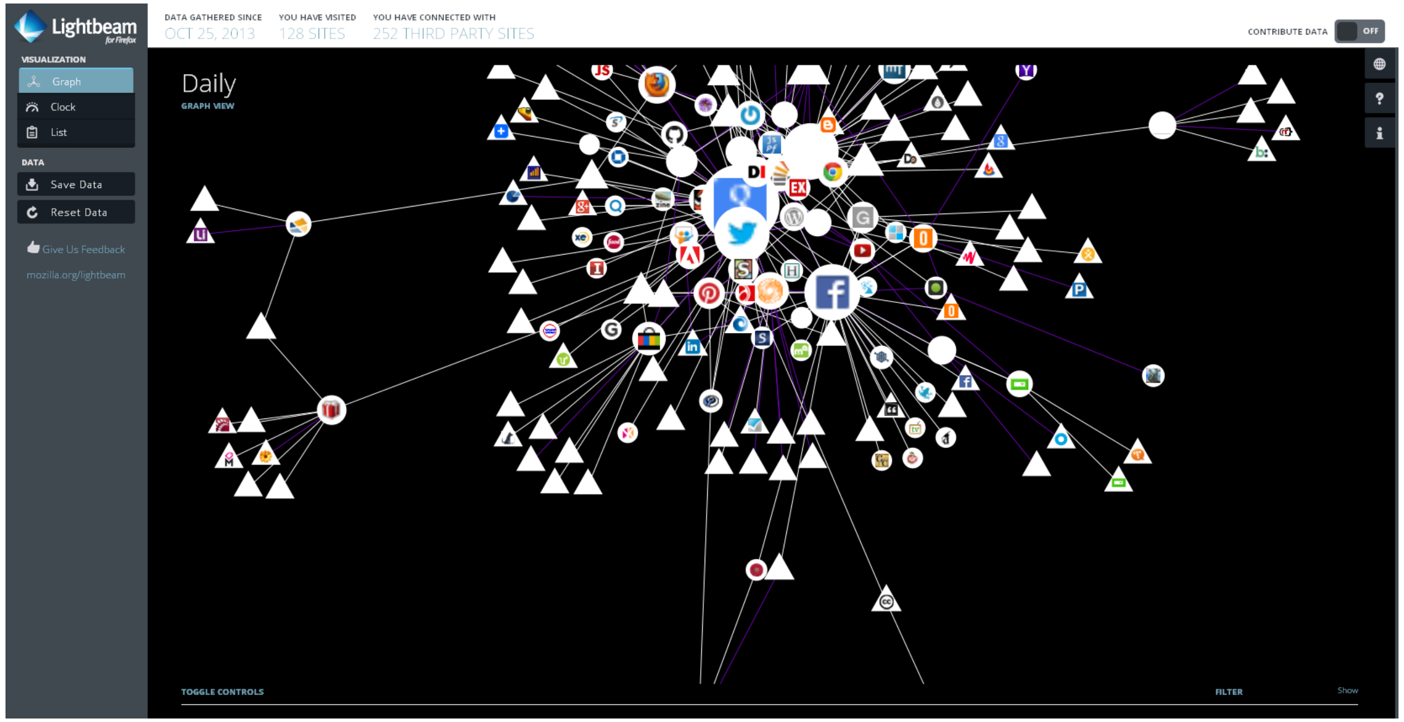
# Intermezzo: Cookie-based Tracking

- Tracking by means of cookies uses the same mechanisms



# Intermezzo: Cookie-based Tracking

- Tracking by means of cookies uses the same mechanisms
  - Page context is stored alongside with your identifier
  - Information from multiple sites is linked together
  - Once you're on an authenticated page, they have even more info



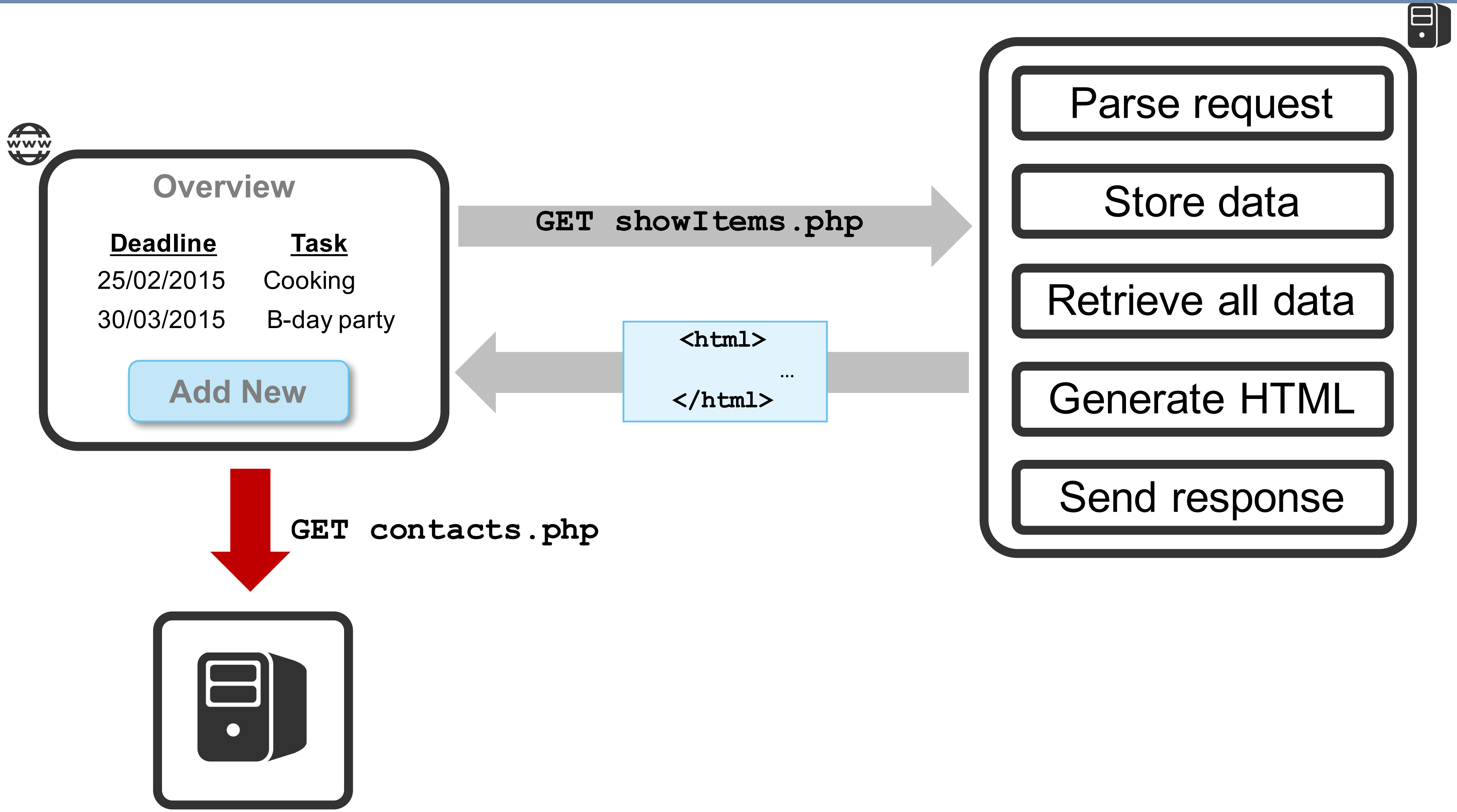
https://support.mozilla.org/en-US/questions/975325?page=2

## Intermezzo: Cookie-based Tracking

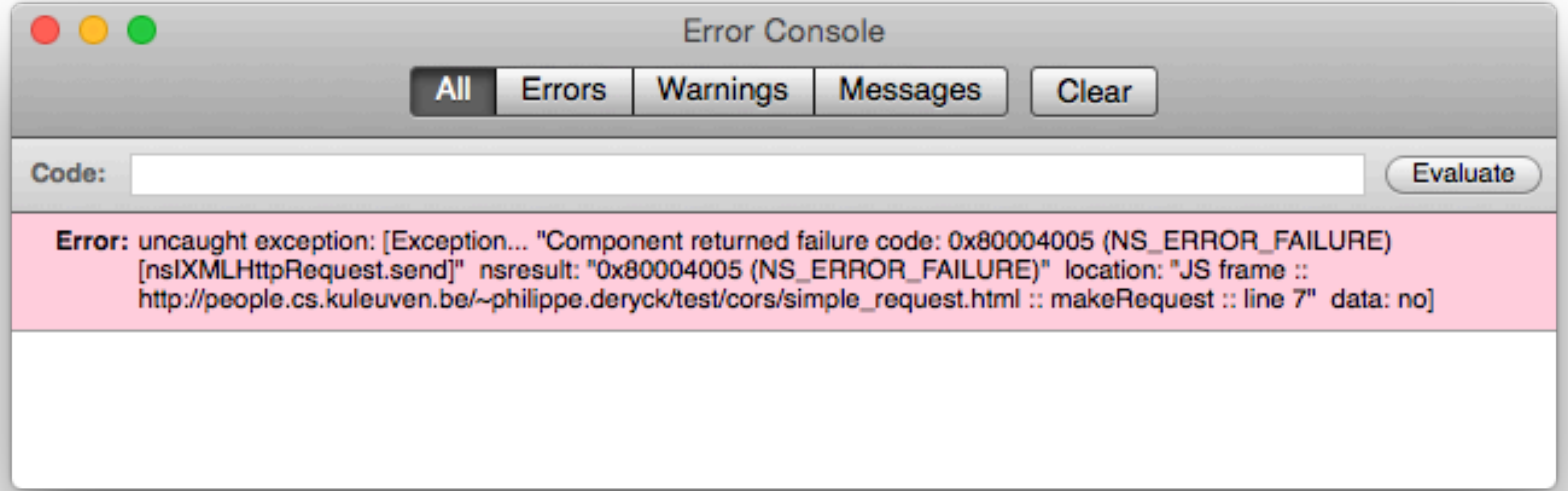
- Tracking by means of cookies uses the same mechanisms
  - Page context is stored alongside with your identifier
  - Information from multiple sites is linked together
  - Once you're on an authenticated page, they have even more info
- Many attempts to stop this behavior of cookies
  - With little to no success
  - Even the “Do-Not-Track” header is used to compile fingerprints
- Do not worry, there are plenty of tracking mechanisms
  - Browser fingerprinting
  - HTML5 Battery API

## Accessing Cross-Origin APIs

## Web Applications Did not Rely on APIs



## Because Browsers Did not Allow It ...



```
Error: uncaught exception: [Exception... "Component returned failure code: 0x80004005 (NS_ERROR_FAILURE)
       [nsIXMLHttpRequest.send]"  nsresult: "0x80004005 (NS_ERROR_FAILURE)"  location: "JS frame ::
       http://people.cs.kuleuven.be/~philippe.deryck/test/cors/simple_request.html :: makeRequest :: line 7"  data: no]
```

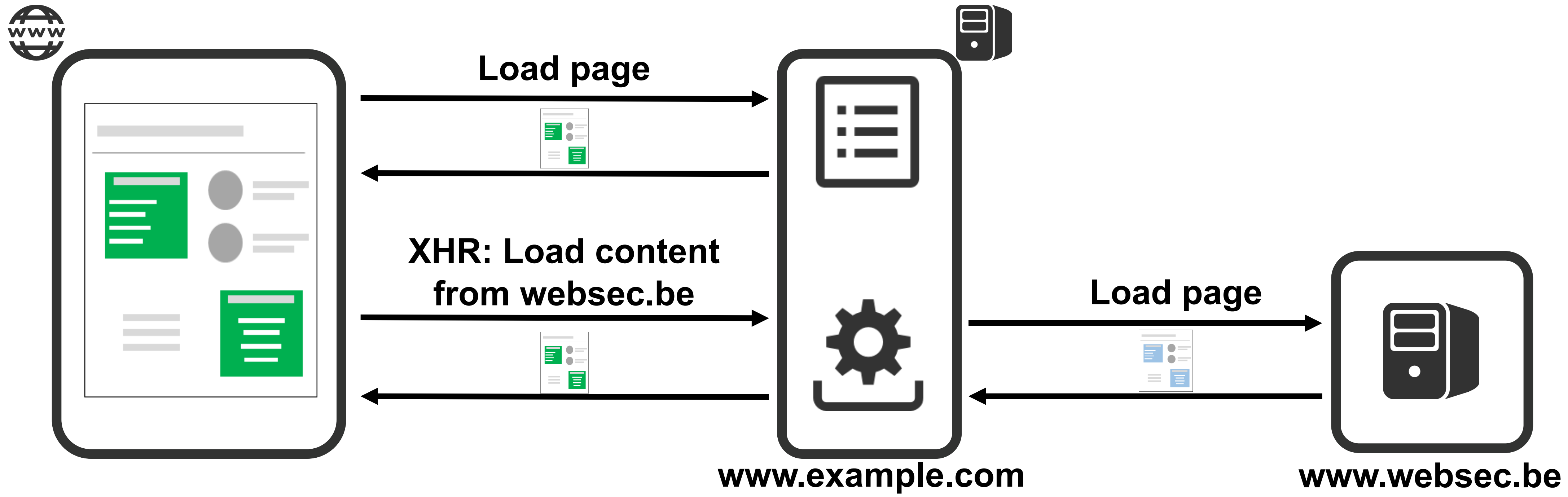
## Cross-Origin Data Access

- Data sharing across applications became a big thing
  - E.g. Google Services, Facebook’s Graph API, …
  - The Web was not built to support these scenarios

- Many useful scenarios sparked creativity, and workarounds
  - Workarounds bypass the browser’s basic security model
  - Often with serious consequences for the security of the site
  - Two commonly used technologies:
    - Server-side proxies
    - JSONP

# Server-Side Proxies

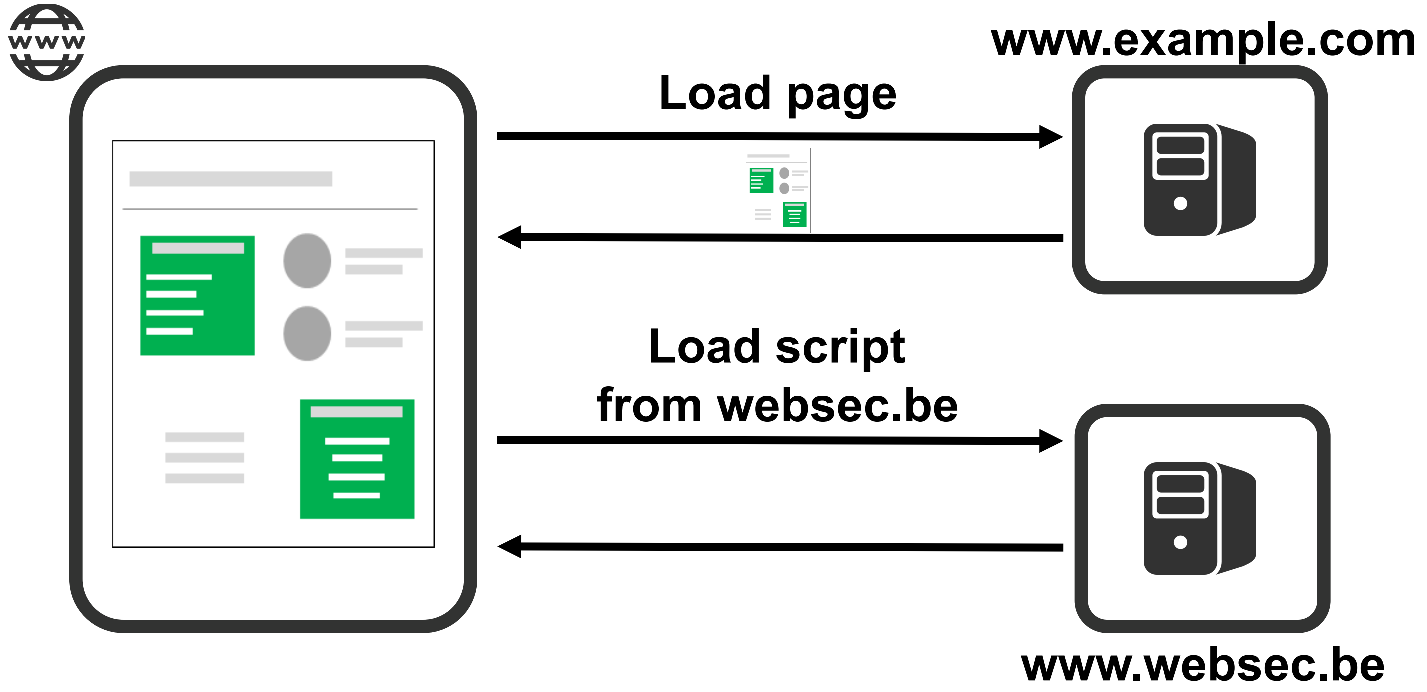
- The browser considers this content to be from the same origin
  - The included data or code runs with the same privileges
  - Script injection by design



## JSON with Padding (JSONP)

- The included data or code runs with the same privileges
  - Script injection by design

```
<script src=“http://www.websec.be/data?callback=showUsers”>
</script>
```



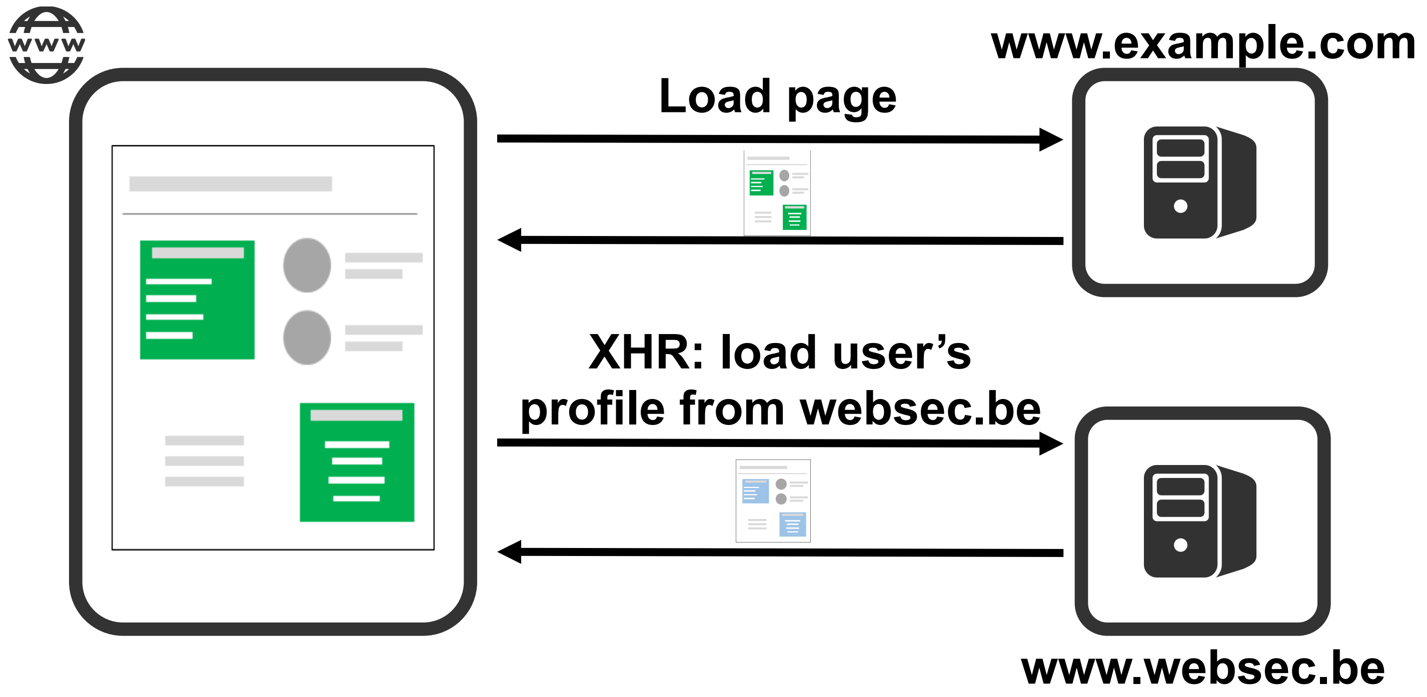
```
showUsers([{"id": 1, "name": "Philippe"},
{"id": 2, "name": "NotPhilippe"}]);
```

## These Workarounds Are Dangerous

- The landscape started to evolve very quickly
  - More and more companies started offering APIs
  - More and more applications started integrating APIs
  - The hacks being used in practice suffer from severe security issues

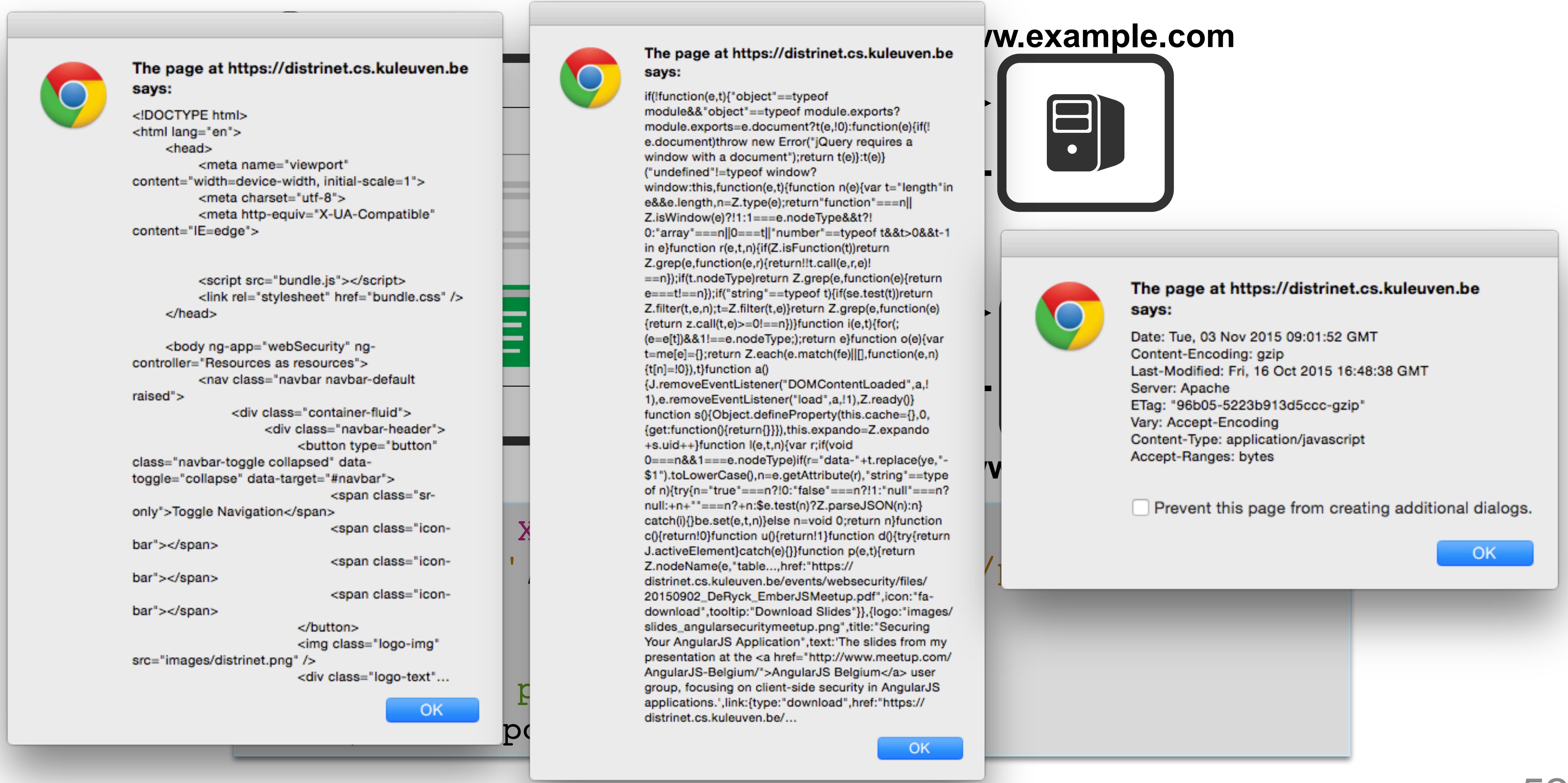
- Essentially, it was time to enable data sharing by design
  - Best approach is to allow XHR to fetch data from another origin
  - This is exactly what Cross-Origin Resource Sharing (CORS) does
    - But it needs 22 pages of specification to do so

# Simply Allowing XHR across Origins



```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'http://www.websec.be/profile', false);
xhr.send();

// Access the profile data
alert(xhr.responseText);
```

# Simply Allowing XHR across Origins



The page at https://distrinet.cs.kuleuven.be says:

```
<!DOCTYPE html>
<html lang="en">
        <head>
                <meta name="viewport"
content="width=device-width, initial-scale=1">
                <meta charset="utf-8">
                <meta http-equiv="X-UA-Compatible"
content="IE=edge">


                <script src="bundle.js"></script>
                <link rel="stylesheet" href="bundle.css" />
        </head>

        <body ng-app="webSecurity" ng-
controller="Resources as resources">
                <nav class="navbar navbar-default
raised">
                        <div class="container-fluid">
                                <div class="navbar-header">
                                        <button type="button"
class="navbar-toggle collapsed" data-
toggle="collapse" data-target="#navbar">
                                                <span class="sr-
only">Toggle Navigation</span>
                                                <span class="icon-
bar"></span>
                                                <span class="icon-
bar"></span>
                                                <span class="icon-
bar"></span>
                                        </button>
                                        <img class="logo-img"
src="images/distrinet.png" />
                                        <div class="logo-text"...
```

OK

The page at https://distrinet.cs.kuleuven.be says:

```
if(!function(e,t){"object"==typeof
module&&"object"==typeof module.exports?
module.exports=e.document?t(e,!0):function(e){if(!
e.document)throw new Error("jQuery requires a
window with a document");return t(e)}:t(e)}
("undefined"!=typeof window?
window:this,function(e,t){function n(e){var t="length"in
e&&e.length,n=Z.type(e);return"function"===n||
Z.isWindow(e)?!1:1===e.nodeType&&t?!
0:"array"===n||0===t||"number"==typeof t&&t>0&&t-1
in e}function r(e,t,n){if(Z.isFunction(t))return
Z.grep(e,function(e,r){return!!t.call(e,r,e)!
==n});if(t.nodeType)return Z.grep(e,function(e){return
e===t!==n});if("string"==typeof t){if(se.test(t))return
Z.filter(t,e,n);t=Z.filter(t,e)}return Z.grep(e,function(e)
{return z.call(t,e)>=0!==n})}function i(e,t){for(;
(e=e[t])&&1!==e.nodeType;);return e}function o(e){var
t=me[e]={};return Z.each(e.match(fe)||[],function(e,n)
{t[n]=!0}),t}function a()
{J.removeEventListener("DOMContentLoaded",a,!
1),e.removeEventListener("load",a,!1),Z.ready()}
function s(){Object.defineProperty(this.cache={},0,
{get:function(){return{}}}),this.expando=Z.expando
+s.uid++}function l(e,t,n){var r;if(void
0===n&&1===e.nodeType)if(r="data-"+t.replace(ye,"-
$1").toLowerCase(),n=e.getAttribute(r),"string"==type
of n){try{n="true"===n?!0:"false"===n?!1:"null"===n?
null:+n+""===n?+n:$e.test(n)?Z.parseJSON(n):n}
catch(i){}be.set(e,t,n)}else n=void 0;return n}function
c(){return!0}function u(){return!1}function d(){try{return
J.activeElement}catch(e){}}function p(e,t){return
Z.nodeName(e,"table...,href:"https://
distrinet.cs.kuleuven.be/events/websecurity/files/
20150902_DeRyck_EmberJSMeetup.pdf",icon:"fa-
download",tooltip:"Download Slides"}},{logo:"images/
slides_angularsecuritymeetup.png",title:"Securing
Your AngularJS Application",text:'The slides from my
presentation at the <a href="http://www.meetup.com/
AngularJS-Belgium/">AngularJS Belgium</a> user
group, focusing on client-side security in AngularJS
applications.',link:{type:"download",href:"https://
distrinet.cs.kuleuven.be/...
```

OK

The page at https://distrinet.cs.kuleuven.be says:

```
Date: Tue, 03 Nov 2015 09:01:52 GMT
Content-Encoding: gzip
Last-Modified: Fri, 16 Oct 2015 16:48:38 GMT
Server: Apache
ETag: "96b05-5223b913d5ccc-gzip"
Vary: Accept-Encoding
Content-Type: application/javascript
Accept-Ranges: bytes
```

☐ Prevent this page from creating additional dialogs.

OK

## Cross-Origin XHR Is also Dangerous!

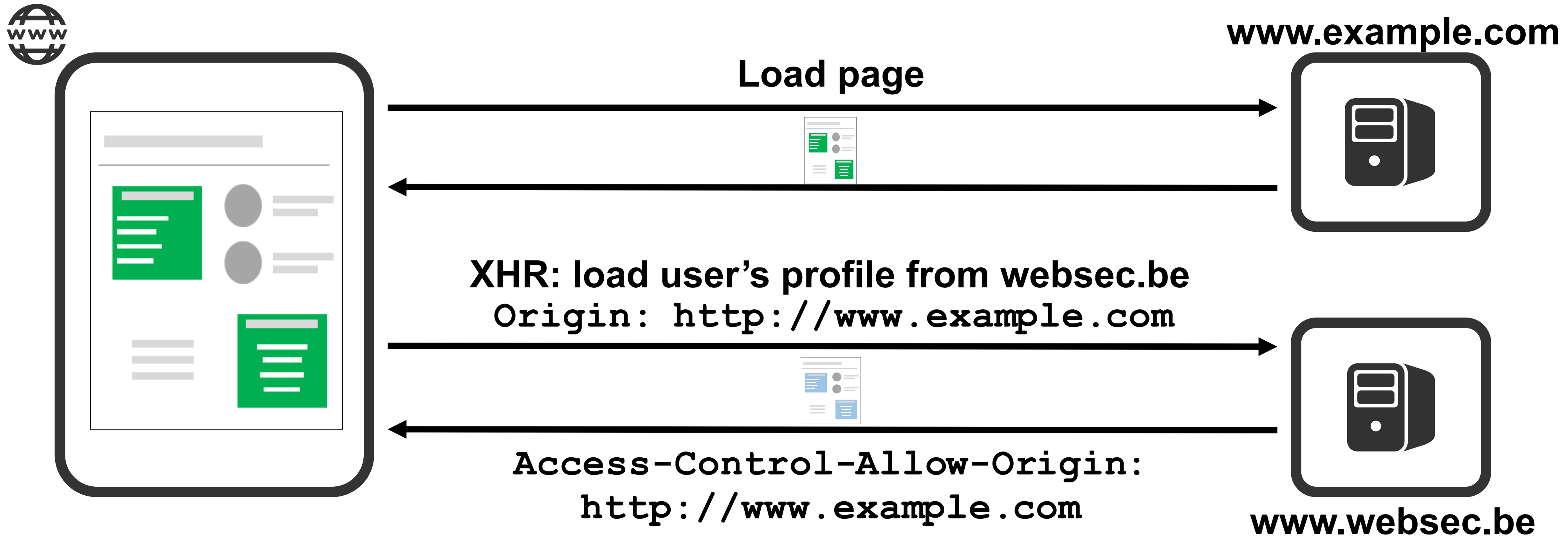
- This would enable scenarios that were previously impossible
  - Not something you want on the Web

- CORS addresses this problem

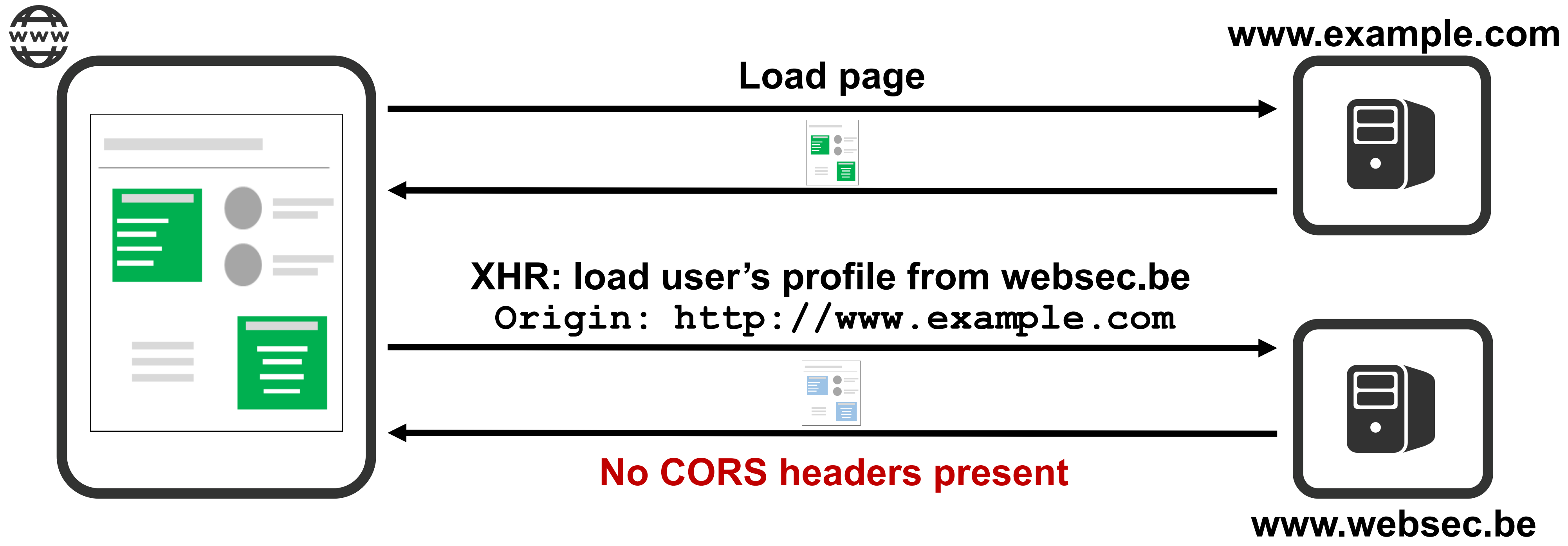
CORS allows the **server** to tell the **browser** whether a **resource** can be accessed by a specific **origin**

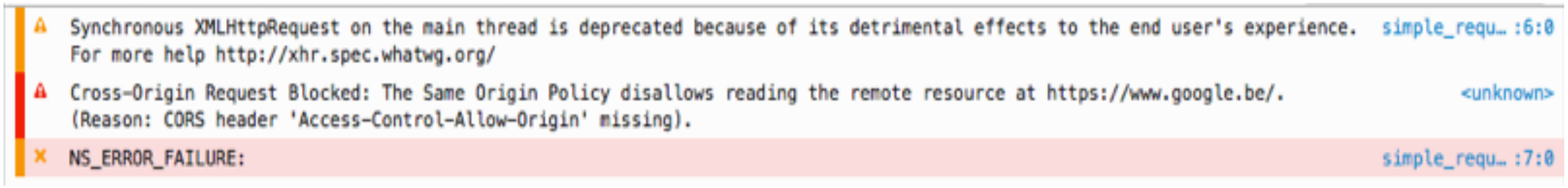
- The browser enforces these checks on the XHR call
  - If everything is OK, access to the resource (response) is granted
  - Otherwise, access is denied

## Simple CORS Example


www.example.com
Load page
XHR: load user's profile from websec.be
Origin: http://www.example.com
Access-Control-Allow-Origin: http://www.example.com
www.websec.be

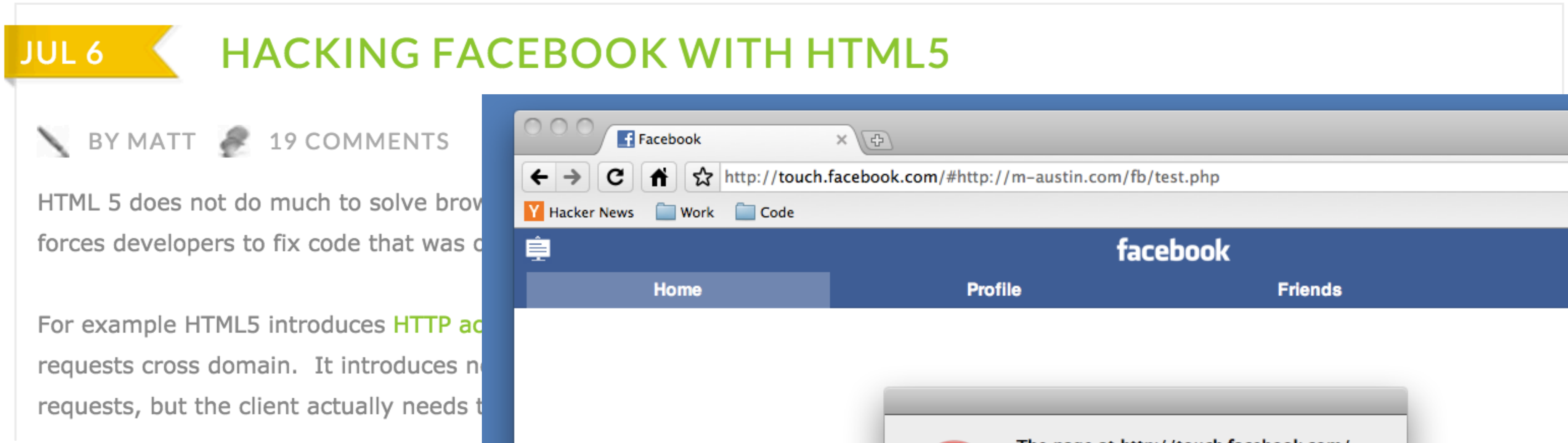
# CORS Protects Legacy Servers by Design


www.example.com
Load page
XHR: load user's profile from websec.be
Origin: http://www.example.com
No CORS headers present
www.websec.be



```
⚠ Synchronous XMLHttpRequest on the main thread is deprecated because of its detrimental effects to the end user's experience.   simple_requ… :6:0
  For more help http://xhr.spec.whatwg.org/
⚠ Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at https://www.google.be/.          <unknown>
  (Reason: CORS header 'Access-Control-Allow-Origin' missing).
✕ NS_ERROR_FAILURE:                                                                                                            simple_requ… :7:0
```

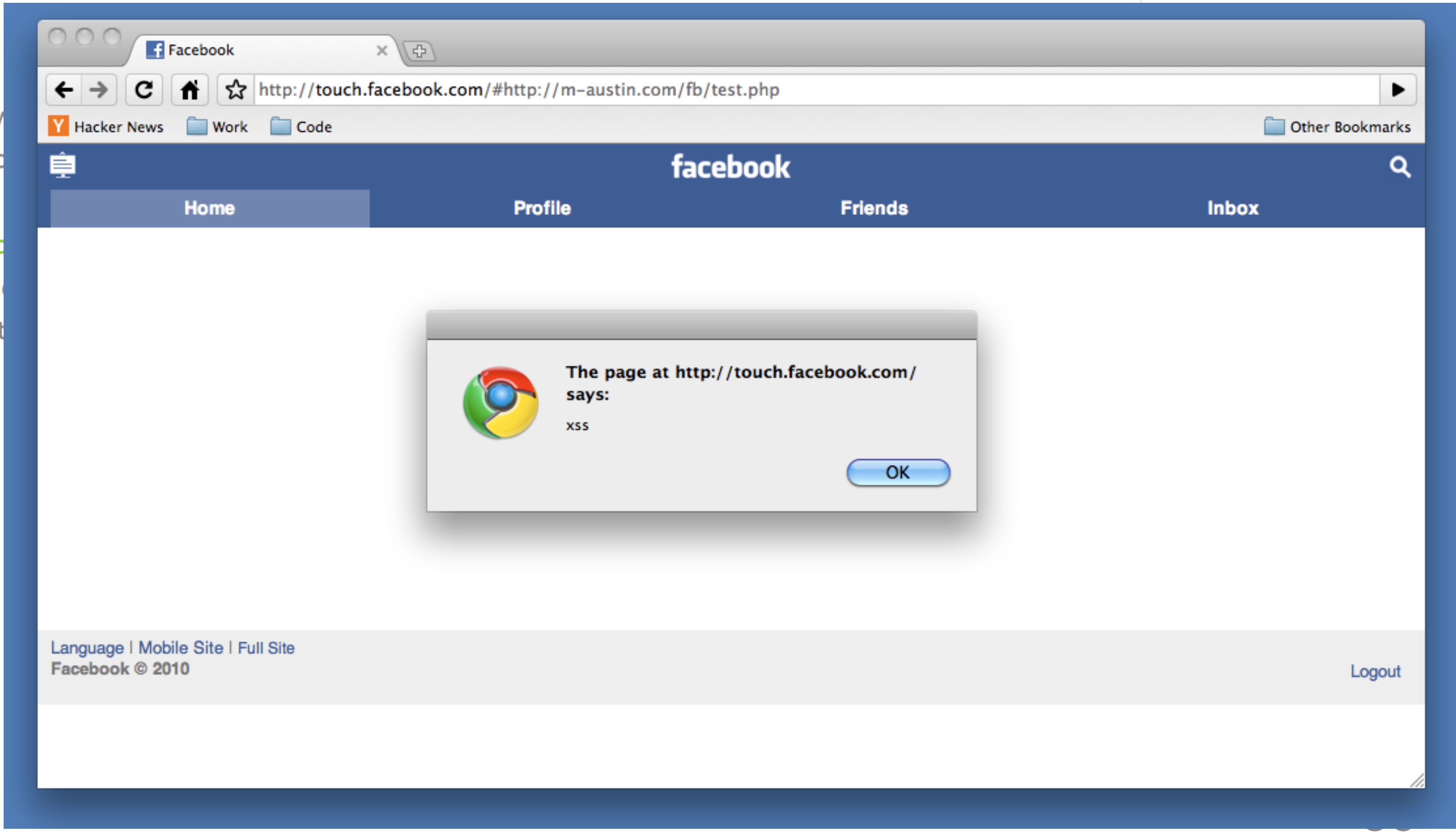
## Violates Implicit Client-Side Security Assumption


JUL 6 HACKING FACEBOOK WITH HTML5

BY MATT 19 COMMENTS

HTML 5 does not do much to solve brow
forces developers to fix code that was c

For example HTML5 introduces HTTP ac
requests cross domain. It introduces n
requests, but the client actually needs t



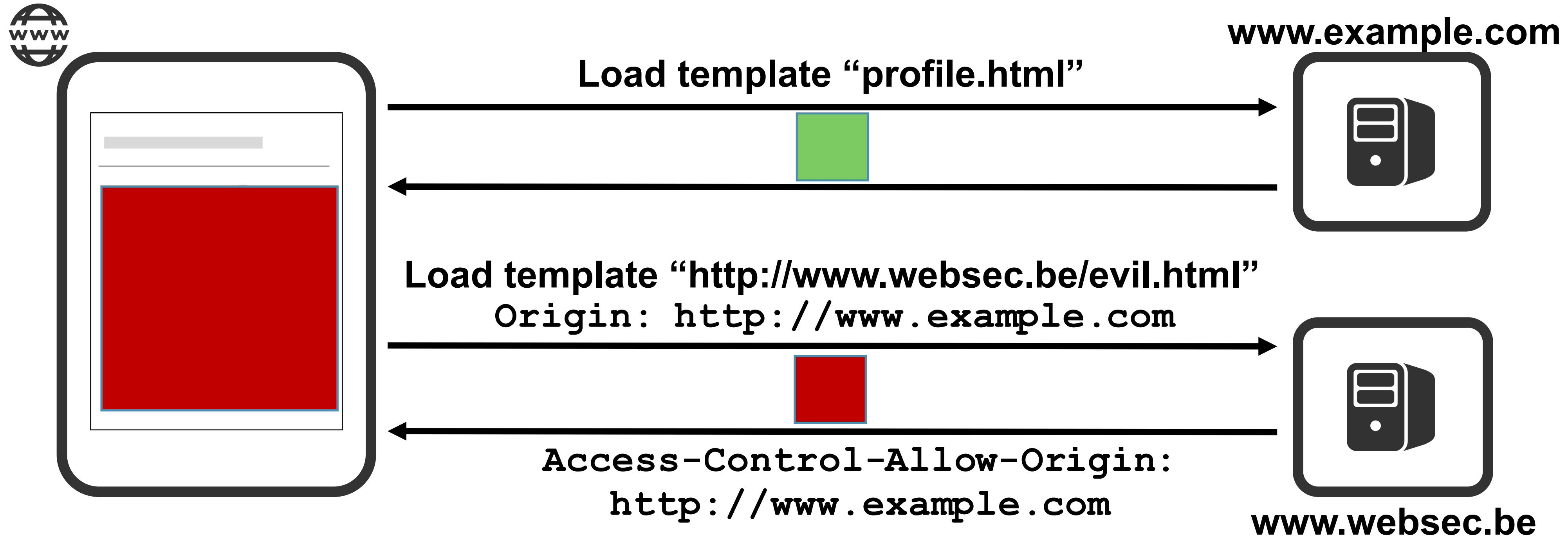
The page at http://touch.facebook.com/ says:

xss

*http://m-austin.com/blog/?p=19*

## Violates Implicit Client-Side Security Assumption

XHR requests can only be sent to the application's origin


www.example.com
Load template "profile.html"
Load template "http://www.websec.be/evil.html"
Origin: http://www.example.com
Access-Control-Allow-Origin:
http://www.example.com
www.websec.be

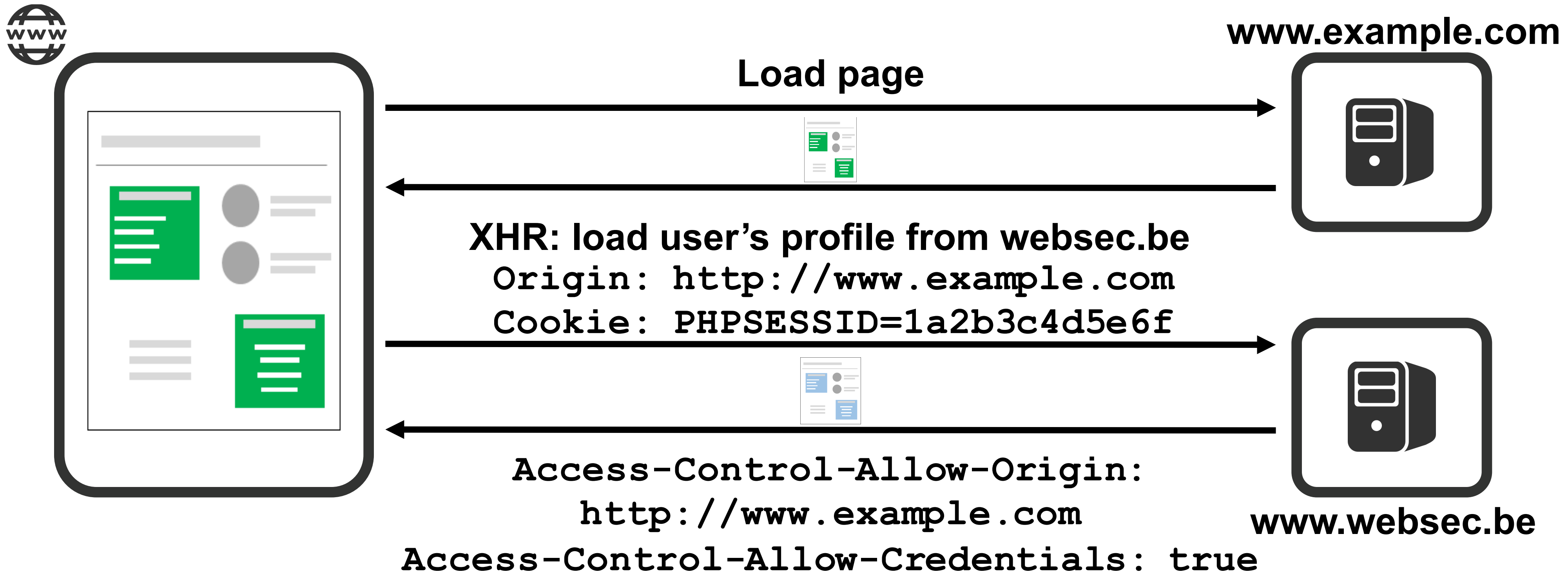
```
function onNavigate(page) {
  var xhr = new XMLHttpRequest();
  xhr.open("GET", page, false);
  xhr.send();
  //inject response into the page
}
```

## Handling Credentials

- Requests can be anonymous or authenticated
  - By default, credentials (i.e. cookies) are not sent
  - Can be enabled by setting the withCredentials flag
- When credentials are used, the server must acknowledge this
  - By sending the Access-Control-Allow-Credentials response header
- Aim is to prevent illegitimate use of the user's credentials
  - Not intended to protect the server from malicious requests

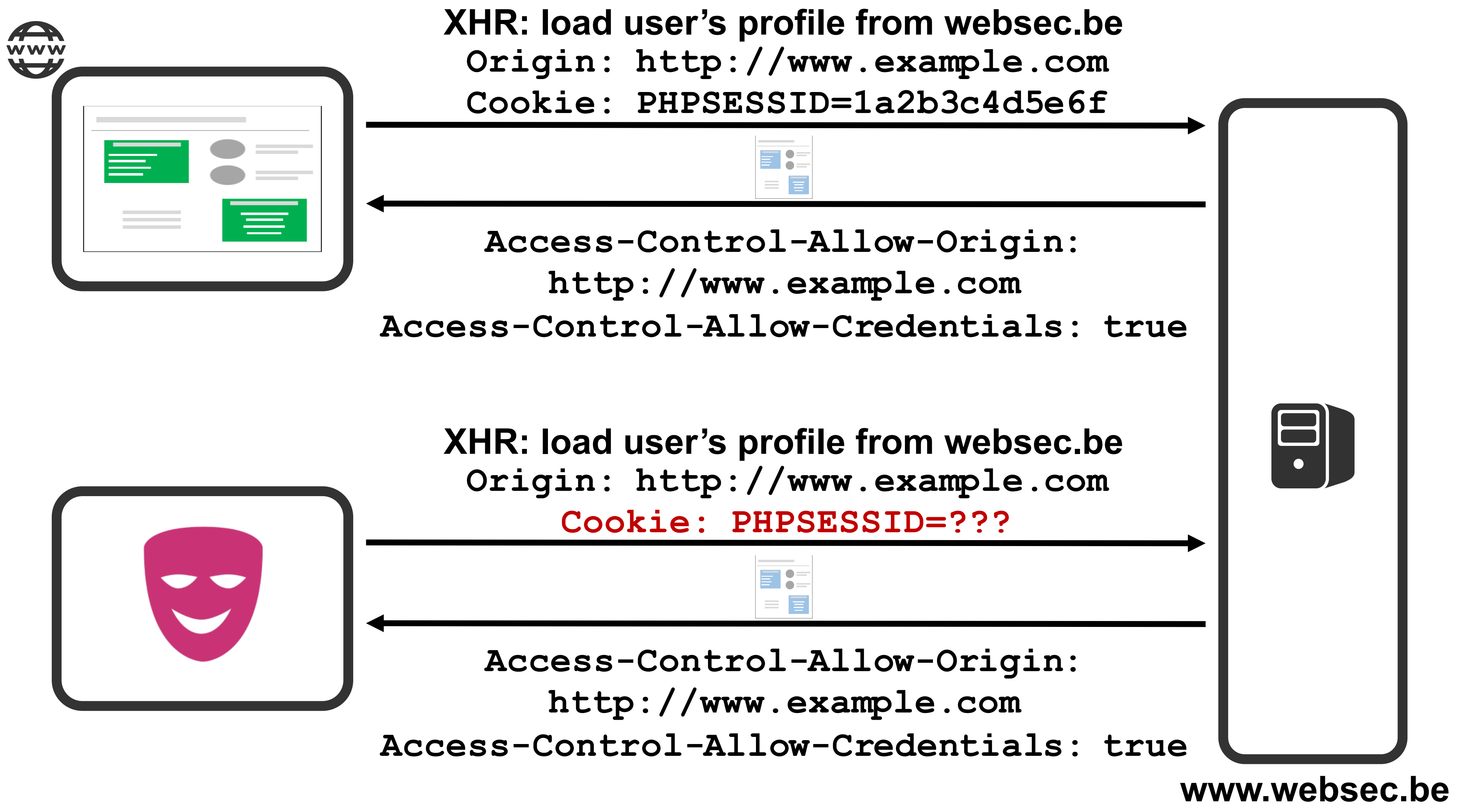
## Simple CORS Example with Credentials

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'http://www.websec.be/profile', false);
xhr.withCredentials = true;
xhr.send();
```



www.example.com
Load page
XHR: load user's profile from websec.be
Origin: http://www.example.com
Cookie: PHPSESSID=1a2b3c4d5e6f
Access-Control-Allow-Origin: http://www.example.com
Access-Control-Allow-Credentials: true
www.websec.be

## Simple CORS Example with Credentials



XHR: load user's profile from websec.be

```
Origin: http://www.example.com
Cookie: PHPSESSID=1a2b3c4d5e6f
```

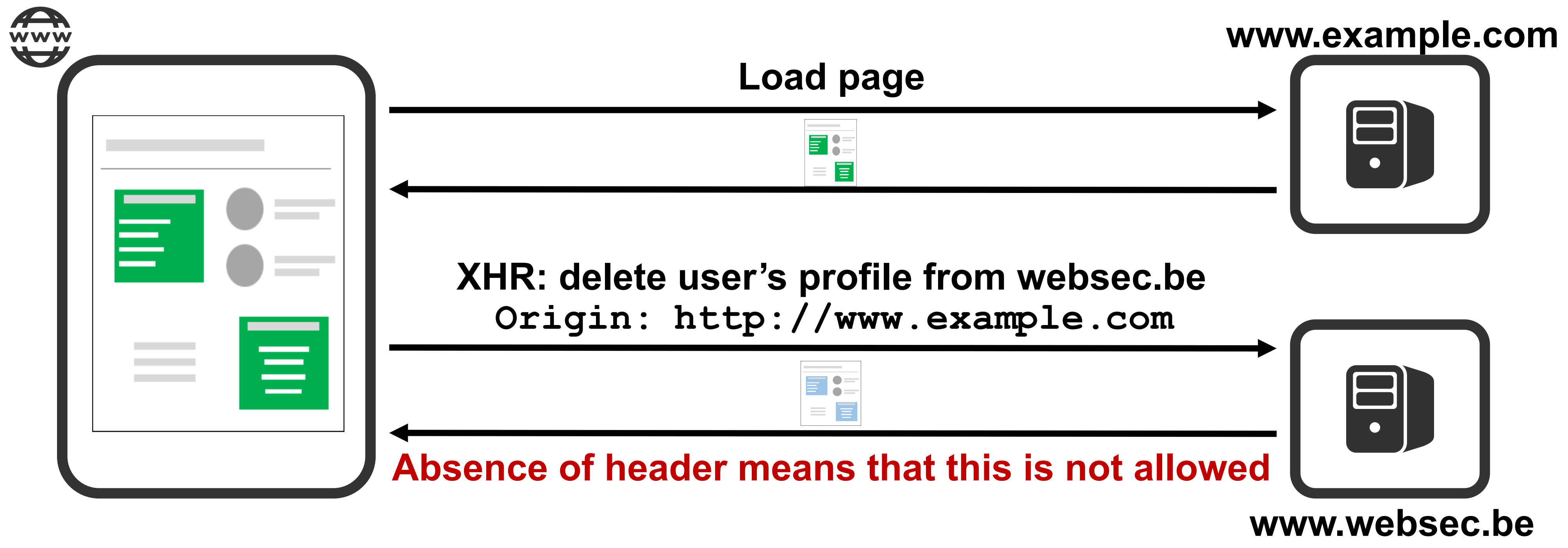
```
Access-Control-Allow-Origin:
http://www.example.com
Access-Control-Allow-Credentials: true
```

XHR: load user's profile from websec.be

```
Origin: http://www.example.com
Cookie: PHPSESSID=???
```

```
Access-Control-Allow-Origin:
http://www.example.com
Access-Control-Allow-Credentials: true
```

# But There Is More ...



www.example.com
Load page
XHR: delete user's profile from websec.be
Origin: http://www.example.com
Absence of header means that this is not allowed
www.websec.be

```
var xhr = new XMLHttpRequest();
xhr.open('DELETE', 'http://www.websec.be/profile/1', false);
xhr.send();
```

## But There Is More ...

- Denying access to the response protects sensitive data
  - Prevents an attacker from reading the user’s data
  - Only possible because data retrieval is a stateless operation
- Most APIs also offer operations that trigger state changes
  - Creating, updating and deleting resources
  - Denying access to the response does not really cut it here …
- CORS requires explicit approval for these requests

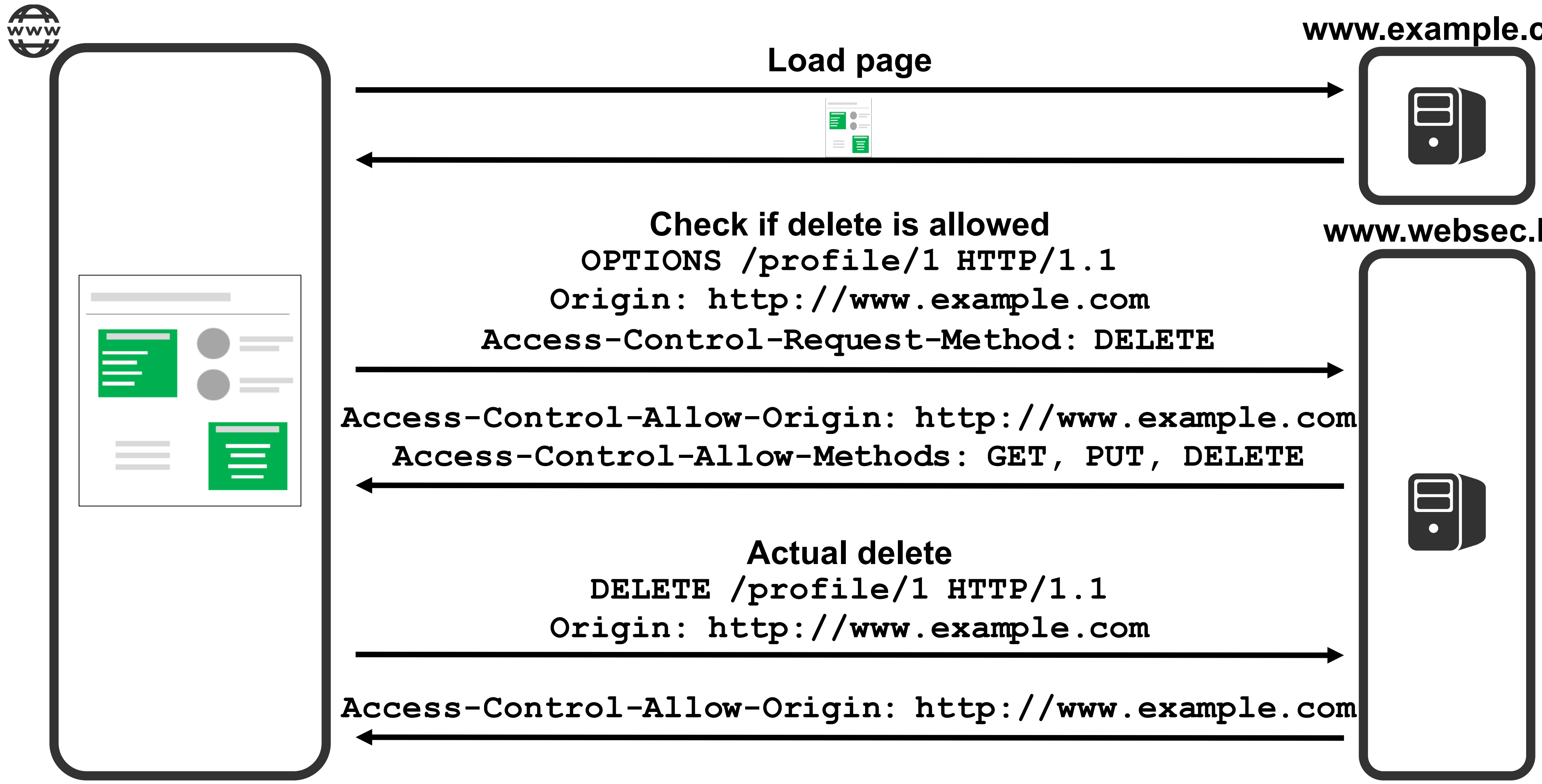
## Explicitly Approving Requests

- CORS aims to prevent giving the attacker more capabilities
  - Makes a distinction between simple and non-simple requests

- Simple requests were already possible before CORS
  - A GET or a simple POST across origins was always possible
  - E.g. fetching an image from another origin
  - E.g. submitting a form to another origin

- Non-simple requests were not possible before CORS
  - E.g. a cross-origin DELETE or PUT request
  - E.g. attaching additional headers to a cross-origin POST request

## Non-Simple Requests Require a Preflight



Check if delete is allowed

```
OPTIONS /profile/1 HTTP/1.1
Origin: http://www.example.com
Access-Control-Request-Method: DELETE
```

```
Access-Control-Allow-Origin: http://www.example.com
Access-Control-Allow-Methods: GET, PUT, DELETE
```

Actual delete

```
DELETE /profile/1 HTTP/1.1
Origin: http://www.example.com
```

```
Access-Control-Allow-Origin: http://www.example.com
```

## Preflight Configuration Options

- The preflight request asks for permission
  - Based on the data that will be sent with the actual request
  - Is sent automatically by the browser for non-simple requests

- CORS preflight request headers
  - Origin
  - Access-Control-Request-Method
    - Specify the method that will be used for the non-simple request
  - Access-Control-Request-Headers
    - Specify which custom headers will be added to the request

## Preflight Configuration Options

- CORS response headers on preflight requests
  - Access-Control-Allow-Origin
  - Access-Control-Allow-Methods
    - Indicate which methods can be used for the actual request
  - Access-Control-Allow-Headers
    - Indicate which request headers can be used for the actual request
  - Access-Control-Max-Age
    - Indicate how long this response should be cached
  - Access-Control-Allow-Credentials
    - Indicate if the actual request can include credentials

## CORS Request and Response Headers

- CORS request headers
  - Origin

- CORS response headers on actual requests
  - Access-Control-Allow-Origin
  - Access-Control-Expose-Headers
    - Indicate which response headers the browser can expose
  - Access-Control-Allow-Credentials
    - Indicate if the response can be exposed if credentials are used

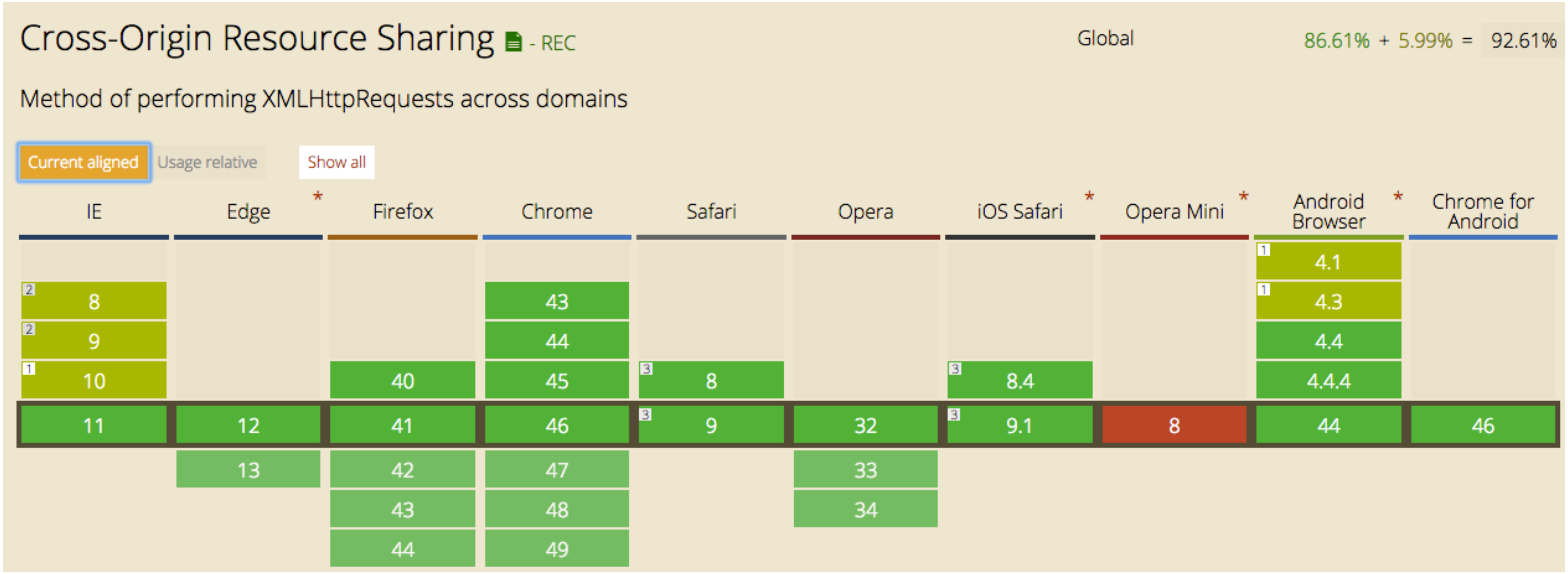
## Preflight Requests in Practice

- Server configuration
  - Developer needs to specify a policy
  - Plenty of frameworks/middleware offer automatic configuration

- For efficiency reasons, preflights are cached by the browser
  - Eliminates an additional round trip for the lifetime of the entry

- Mandatory if you use custom headers (e.g. JWT tokens)
  - CORS only allows cookies without preflight for simple requests

# CORS Support in Browsers



Cross-Origin Resource Sharing - REC

Global 86.61% + 5.99% = 92.61%

Method of performing XMLHttpRequests across domains

Current aligned | Usage relative | Show all

| IE | Edge * | Firefox | Chrome | Safari | Opera | iOS Safari * | Opera Mini * | Android Browser * | Chrome for Android |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | [1] 4.1 | |
| [2] 8 | | | 43 | | | | | [1] 4.3 | |
| [2] 9 | | | 44 | | | | | 4.4 | |
| [1] 10 | | 40 | 45 | [3] 8 | | [3] 8.4 | | 4.4.4 | |
| 11 | 12 | 41 | 46 | [3] 9 | 32 | [3] 9.1 | 8 | 44 | 46 |
| | 13 | 42 | 47 | | 33 | | | | |
| | | 43 | 48 | | 34 | | | | |
| | | 44 | 49 | | | | | | |

http://caniuse.com/#search=cors

## CORS in Practice

- CORS is well supported
  - Virtually all modern browsers are CORS-enabled
  - Many server-side frameworks support CORS configuration
  - Many publicly available APIs already send CORS headers

And many more …

- CORS should be even more widely supported
  - Many new technologies depend on CORS under the hood
  - Mainly to prevent new capabilities from harming legacy servers
  - E.g. HTML5 canvas, Subresource Integrity, …

## Guidelines for Building a CORS Policy

- Private resources should not be accessible with CORS
  - Files that only your application should use, and nobody else
  - Sensitive resources and APIs

```
Do not send any CORS headers in the response
```

- Public resources should be easily accessible with CORS
  - Files that anybody can use, without authorization
  - Published libraries, publicly available images, …
  - Especially non-sensitive, non-private JavaScript files (SRI)

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

# Guidelines for Building a CORS Policy

- Some sensitive, private resources need to be shared
  - E.g. a contact list to be used in a partner application
  - E.g. a button to integrate in other sites, that requires authentication

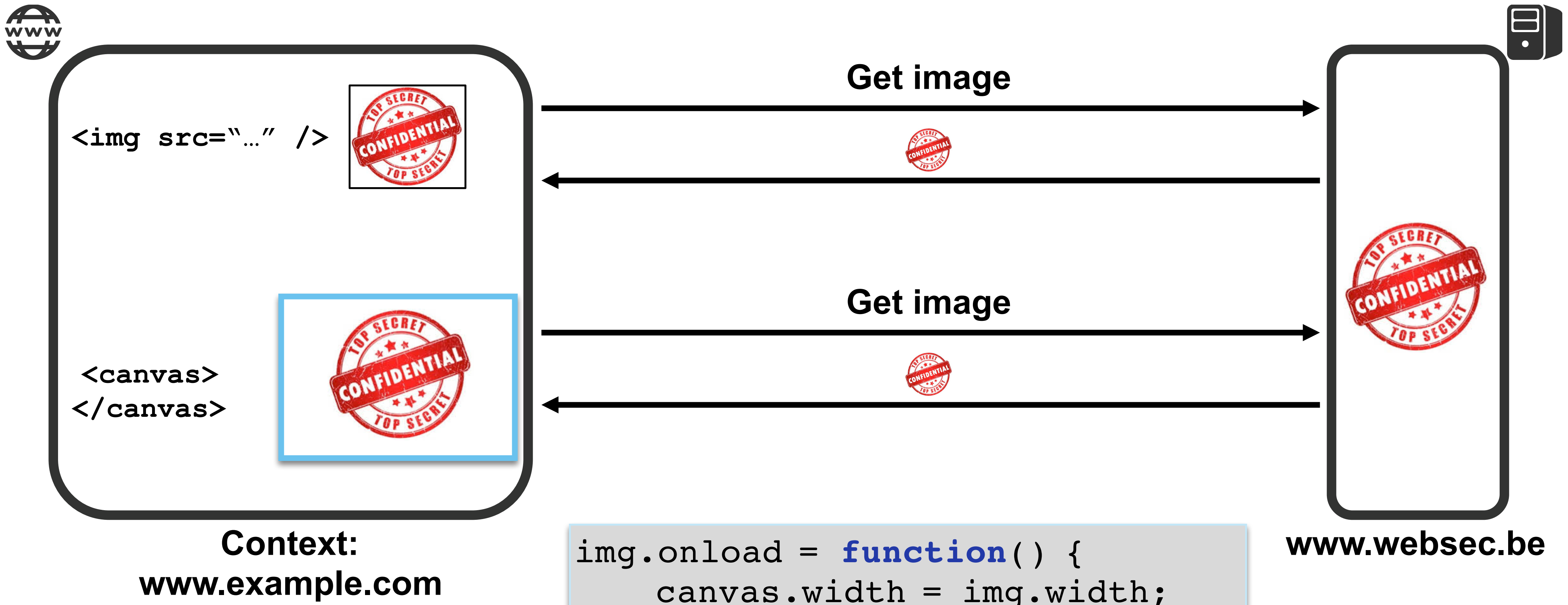
- These steps should be followed by the server
  - Verify if the value of the Origin header matches an accepted origin
    - If not, abort without sending any headers
  - Verify if the user is authorized to access the requested resource
    - Take the method and custom headers into account
    - If not, abort without sending any headers
  - Process the request, and send the appropriate headers

```
Access-Control-Allow-Origin: http://www.example.com
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers: X-API-VERSION
```

## Exciting New Features of a Canvas

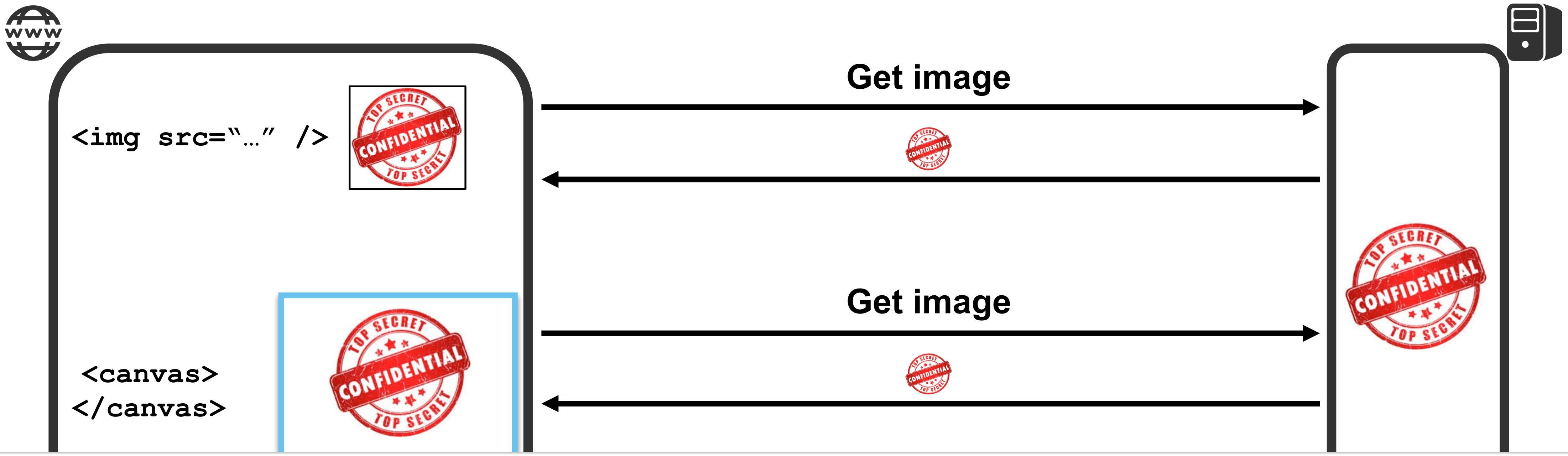
- HTML5 Canvas offers JavaScript-powered graphics
  - Good candidate to replace Flash and similar plugins
  - Enables various impressive scenarios, including entire games
- HTML5 Canvas supports rendering of existing graphics
  - E.g. play an HTML5 video on the surface of a canvas
  - E.g. render an existing image on a canvas and edit it
- Extensive JavaScript API
  - Supports features to inspect images, manipulate images
  - Supports saving as a Data URI or a File blob

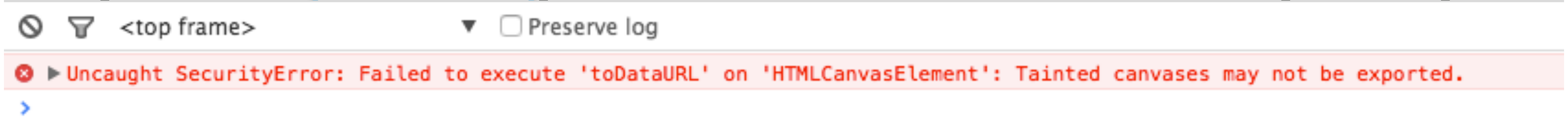
# HTML5 Canvas Enables Image Manipulation



```
img.onload = function() {
    canvas.width = img.width;
    canvas.height = img.height;
    ctx.drawImage( img, 0, 0 );
}
```

# HTML5 Canvas Enables Image Inspection

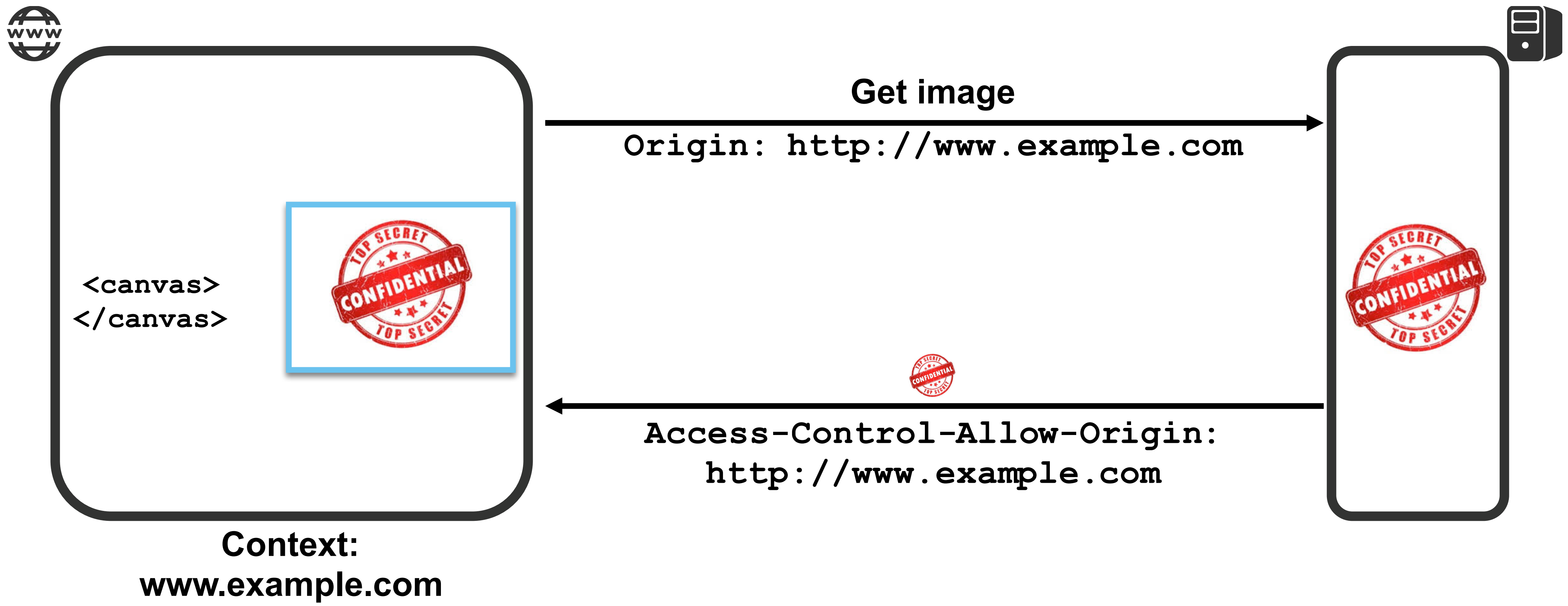




```
Uncaught SecurityError: Failed to execute 'toDataURL' on 'HTMLCanvasElement': Tainted canvases may not be exported.
```

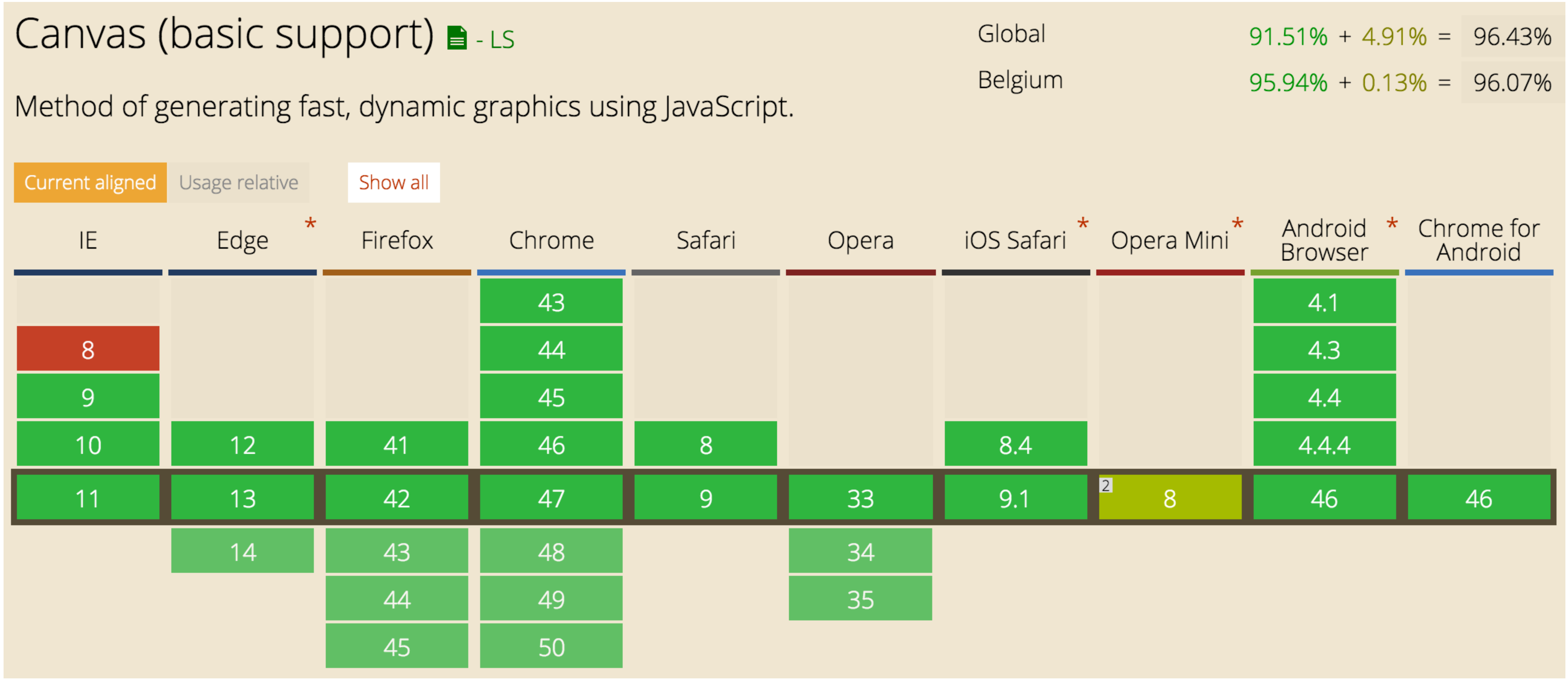
```
// Get the CanvasPixelArray from the given coordinates and dimensions.
var imgd = context.getImageData(x, y, width, height);
var pix = imgd.data;

// Extract as Data URI
canvas.toDataURL("image/png");
```

# Images Can Be Shared Across Origins



```
var img = new Image();
img.crossOrigin = "Anonymous"; //or "use-credentials"
img.src = "http://www.websec.be/topsecret.png";
```

# Canvas Support in Browsers



Canvas (basic support) - LS

Method of generating fast, dynamic graphics using JavaScript.

| | | | | | |
|---|---|---|---|---|---|
| Global | 91.51% | + | 4.91% | = | 96.43% |
| Belgium | 95.94% | + | 0.13% | = | 96.07% |

Current aligned · Usage relative · Show all

| IE | Edge * | Firefox | Chrome | Safari | Opera | iOS Safari * | Opera Mini * | Android Browser * | Chrome for Android |
|---|---|---|---|---|---|---|---|---|---|
| | | | 43 | | | | | 4.1 | |
| 8 | | | 44 | | | | | 4.3 | |
| 9 | | | 45 | | | | | 4.4 | |
| 10 | 12 | 41 | 46 | 8 | | 8.4 | | 4.4.4 | |
| 11 | 13 | 42 | 47 | 9 | 33 | 9.1 | [2] 8 | 46 | 46 |
| | 14 | 43 | 48 | | 34 | | | | |
| | | 44 | 49 | | 35 | | | | |
| | | 45 | 50 | | | | | | |

http://caniuse.com/#search=canvas

## Conclusion

## Embrace the Same-Origin Policy

- The SOP may be old, but it’s still incredibly relevant
  - Context isolation protects your page’s contents
  - Prevents unauthorized access to your origin-based resources

- Direct script inclusion bypasses any origin-based protection
  - Scripts are executed within the page’s context
  - Perfectly fine for your own JavaScript code
  - Very problematic with third-party code

- Protect yourself by combining iframes with new technologies
  - Principle of least privilege with *sandbox* and *CSP*

# Dealing with Remote Communication

- The SOP only protects against direct context interaction
  - Prevents direct access to your session cookies
  - Indirect interactions can still cause unintended consequences
    - Cookies are attached to outgoing requests, causing CSRF

- Scripted remote communication is denied by the SOP
  - Used to be the case before CORS was introduced
  - With CORS, the server is in control over which requests are allowed
  - Complex policy, but very powerful, and already used a lot

## Take Security Seriously

- Isolating content may require a bit more effort
  - But you gain a lot of security guarantees by doing it
- Take inspiration from your predecessors
  - There are plenty more examples besides Dropbox
- Share your experiences
  - And be a leader for others to do the same

# Progressive Web Security course

State-of-the-art technologies

Hands-on labs included

1. Why simply deploying HTTPS will not get you an A+ grade
2. How to avoid common pitfalls in authentication and authorization
3. Why modern security technologies will eradicate XSS
4. Four new browser communication mechanisms, and how they affect you

**3rd edition starts on April 12th 2016**

**https://www.websec.be**

# The Web’s Security Model

## Acknowledgements

Icons by Visual Pharm (https://icons8.com)

# The Web's Security Model

Philippe De Ryck

philippe.deryck@cs.kuleuven.be

@PhilippeDeRyck

/in/philippederyck

https://www.websec.be