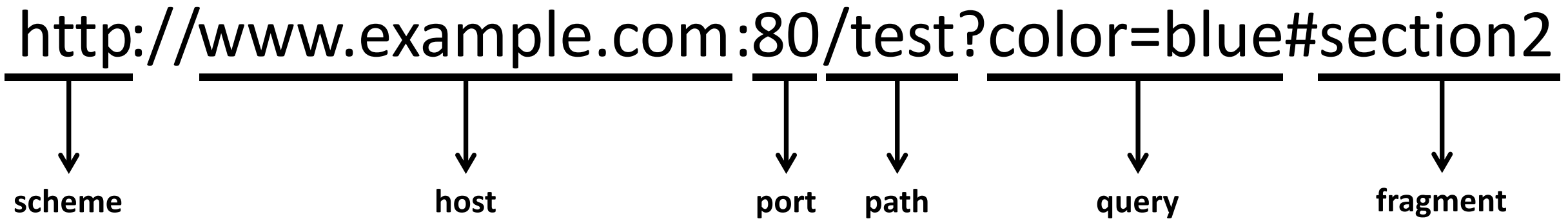
THE SECURITY MODEL OF THE WEB

Philippe De Ryck

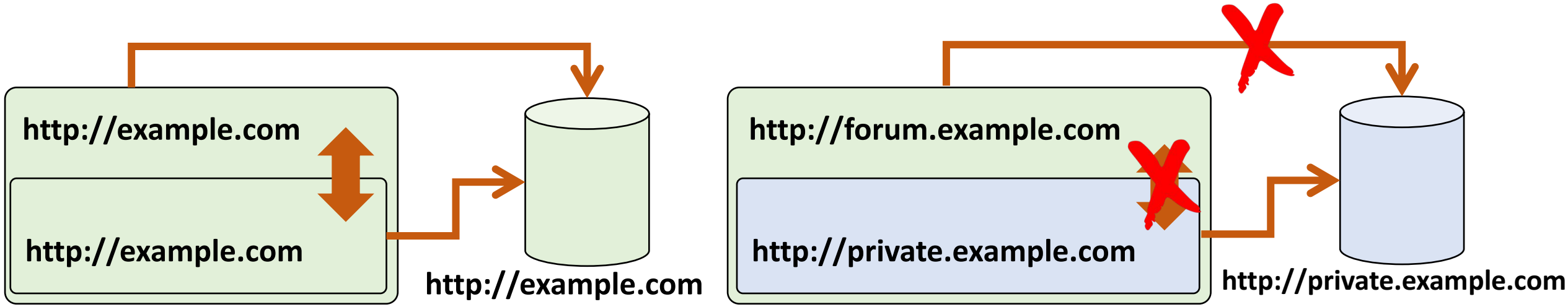
SecAppDev 2018

THE CONCEPT OF AN ORIGIN



Same-Origin Policy (SOP)

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

Why is this so important?

- Understanding the basic security model of the web
  - More and more software is moving towards the web
  - Modern features strongly depend on the Same-Origin Policy
- Web security is an important aspect of SecAppDev
  - Many of the attacks covered this weak abuse the SOP
  - Countermeasures depend on the SOP for their security
- Most security problems are caused by a lack of knowledge
  - If developers are not aware of security problems, they can't fix them

About me – Philippe De Ryck

- My goal is to help you build secure web applications
  - Hosted and customized in-house training
  - Specialized security assessments of modern web applications
  - Threat landscape analysis and prioritization of security efforts

- Specialized in security for Angular applications
  - The security impact of moving to a new paradigm
  - Best practices and guidelines

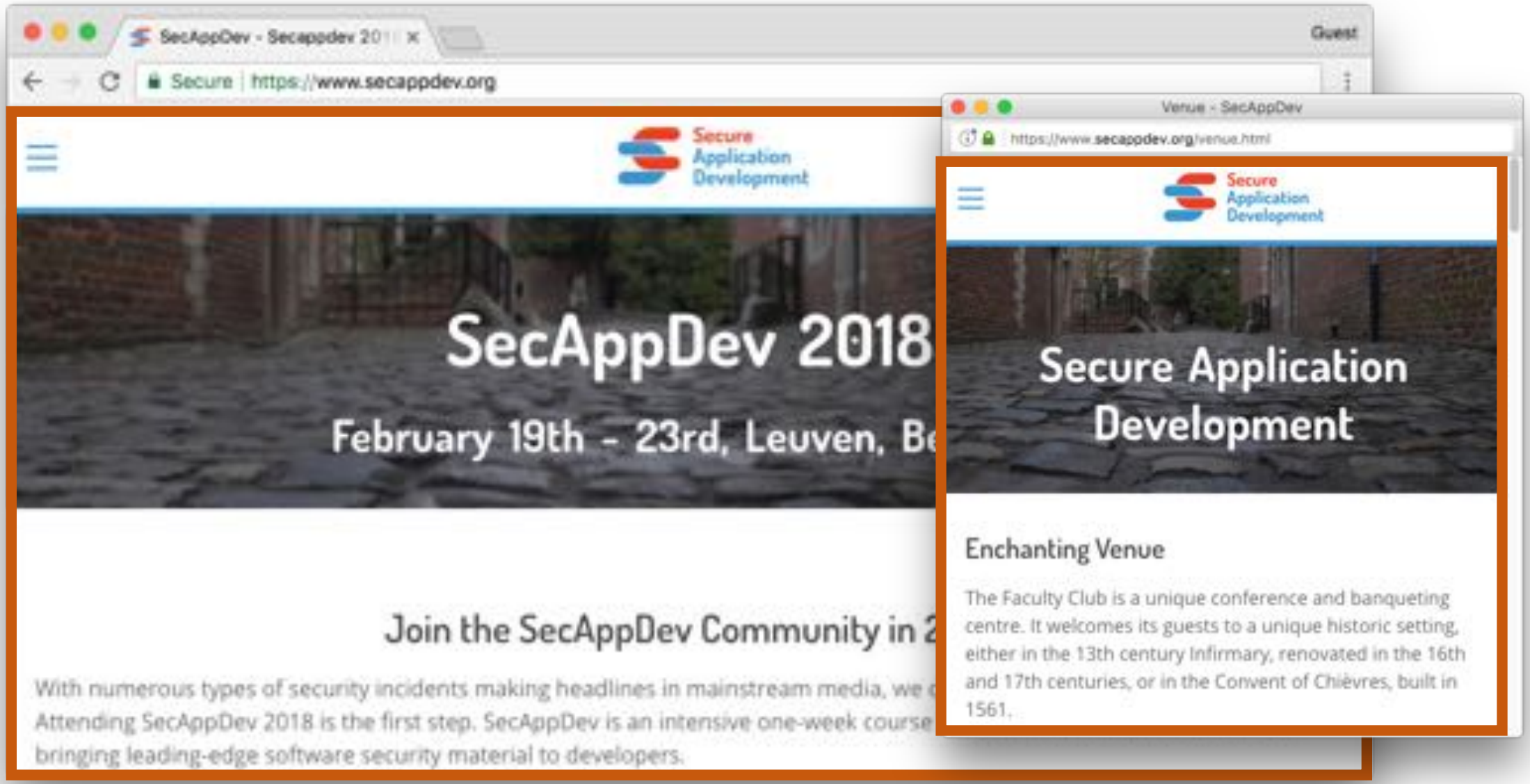
- Course curator and co-organizer of the SecAppDev course
  - Security course targeted towards developers, architects, …
  - Week-long course taught by international experts in their domain

secappdev.org

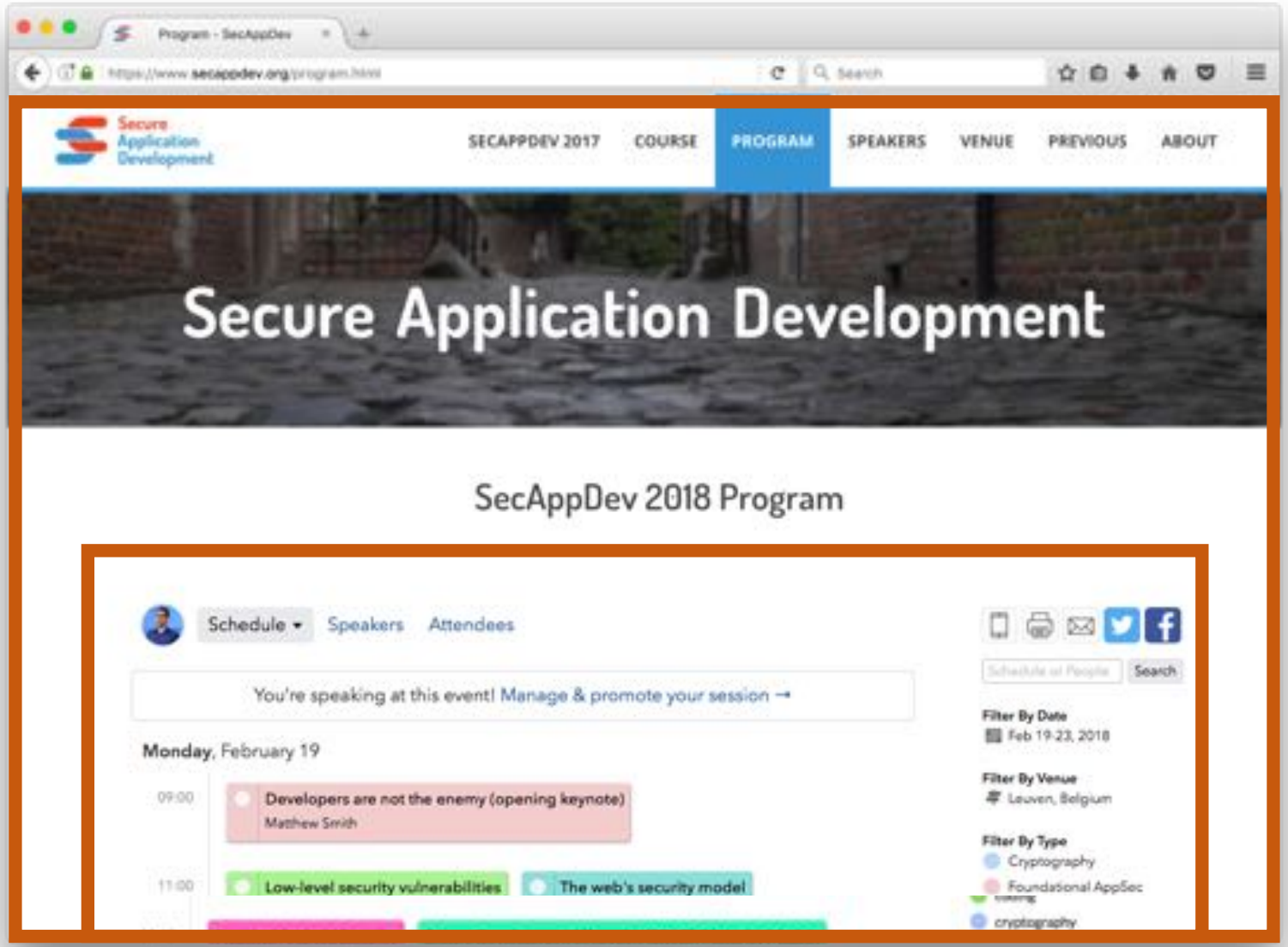
Browsing Contexts

What is a browsing context?

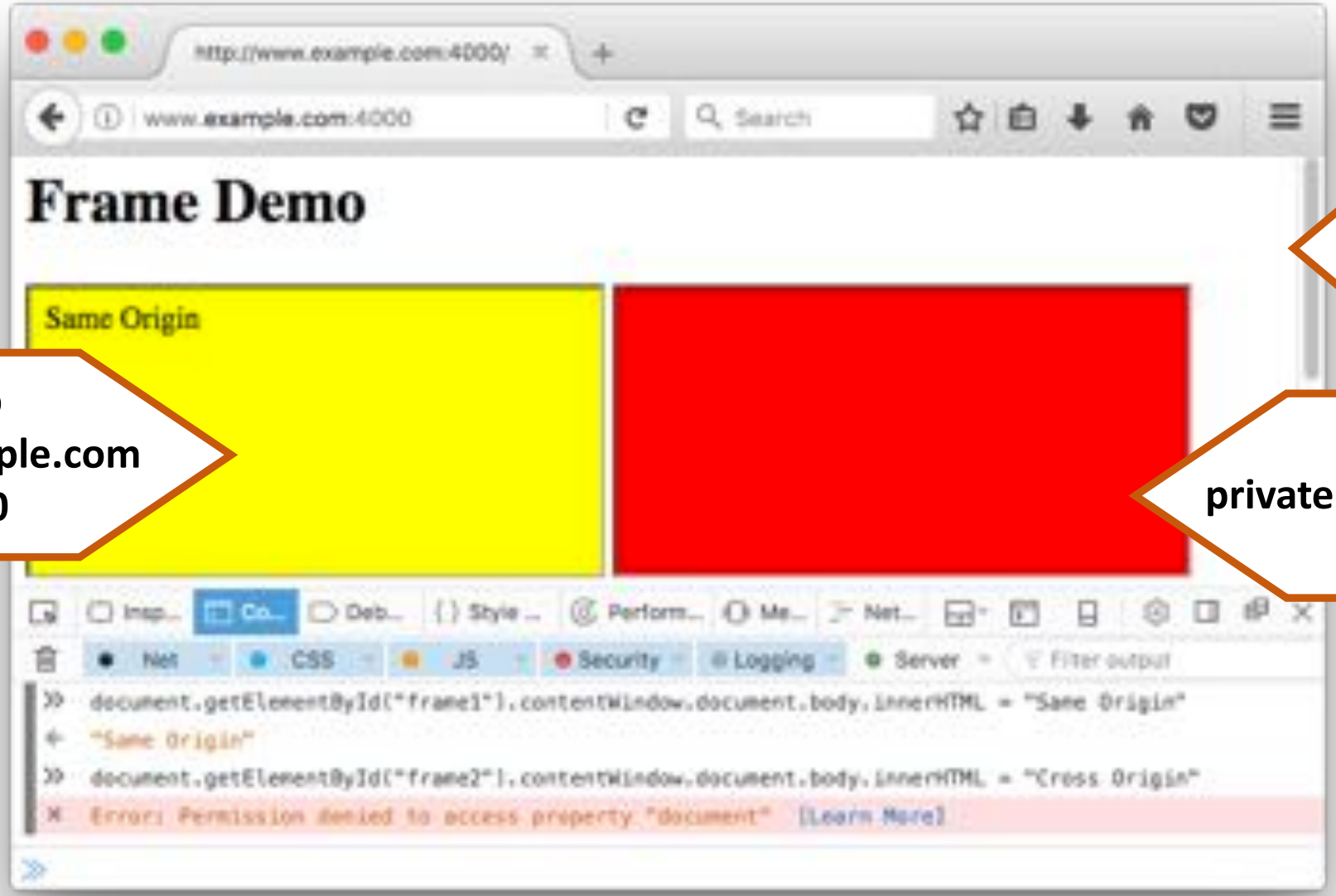
What is a browsing context?



NESTED BROWSING CONTEXTS



THE SOP ISOLATES BROWSING CONTEXTS

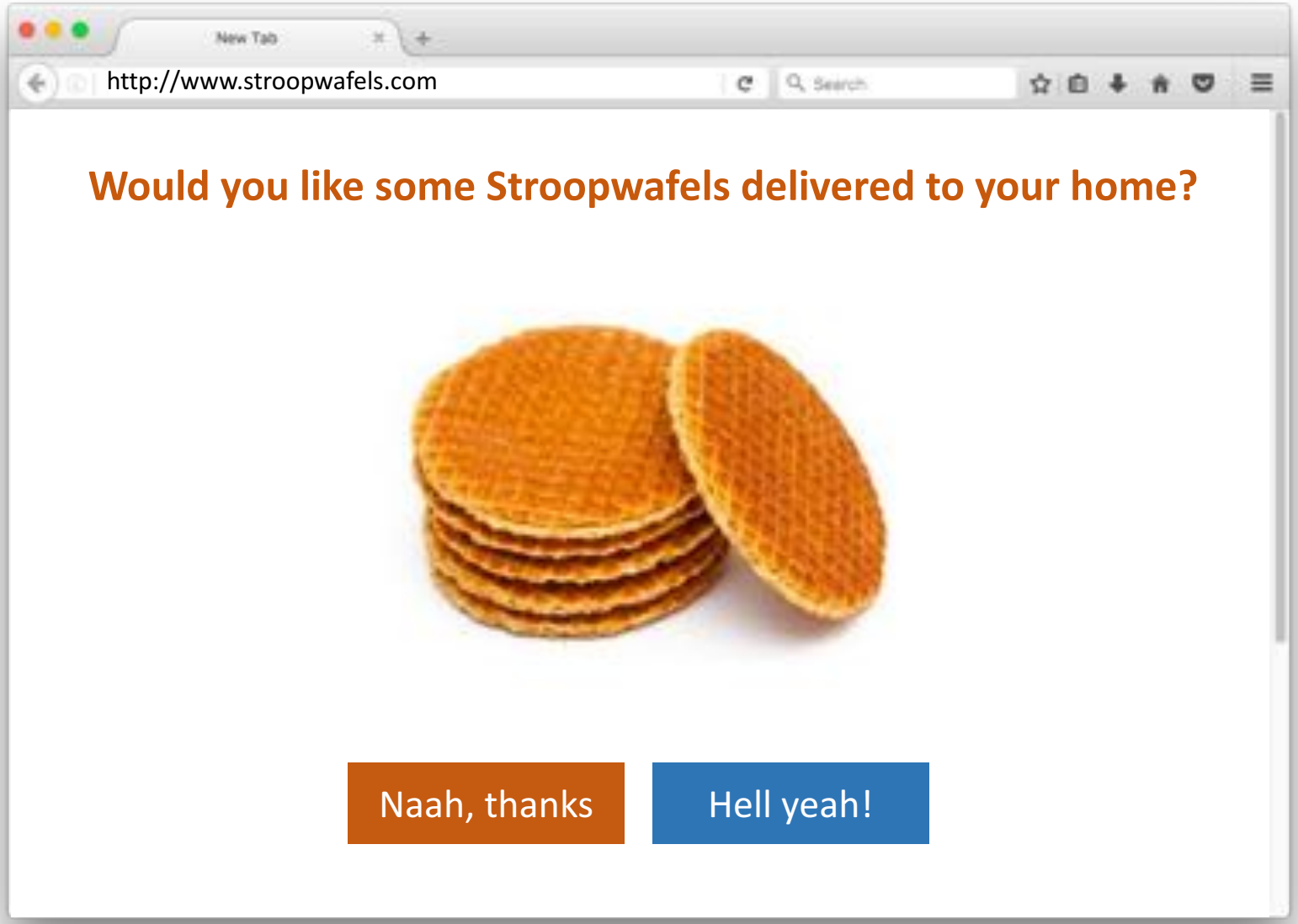


http
www.example.com
4000

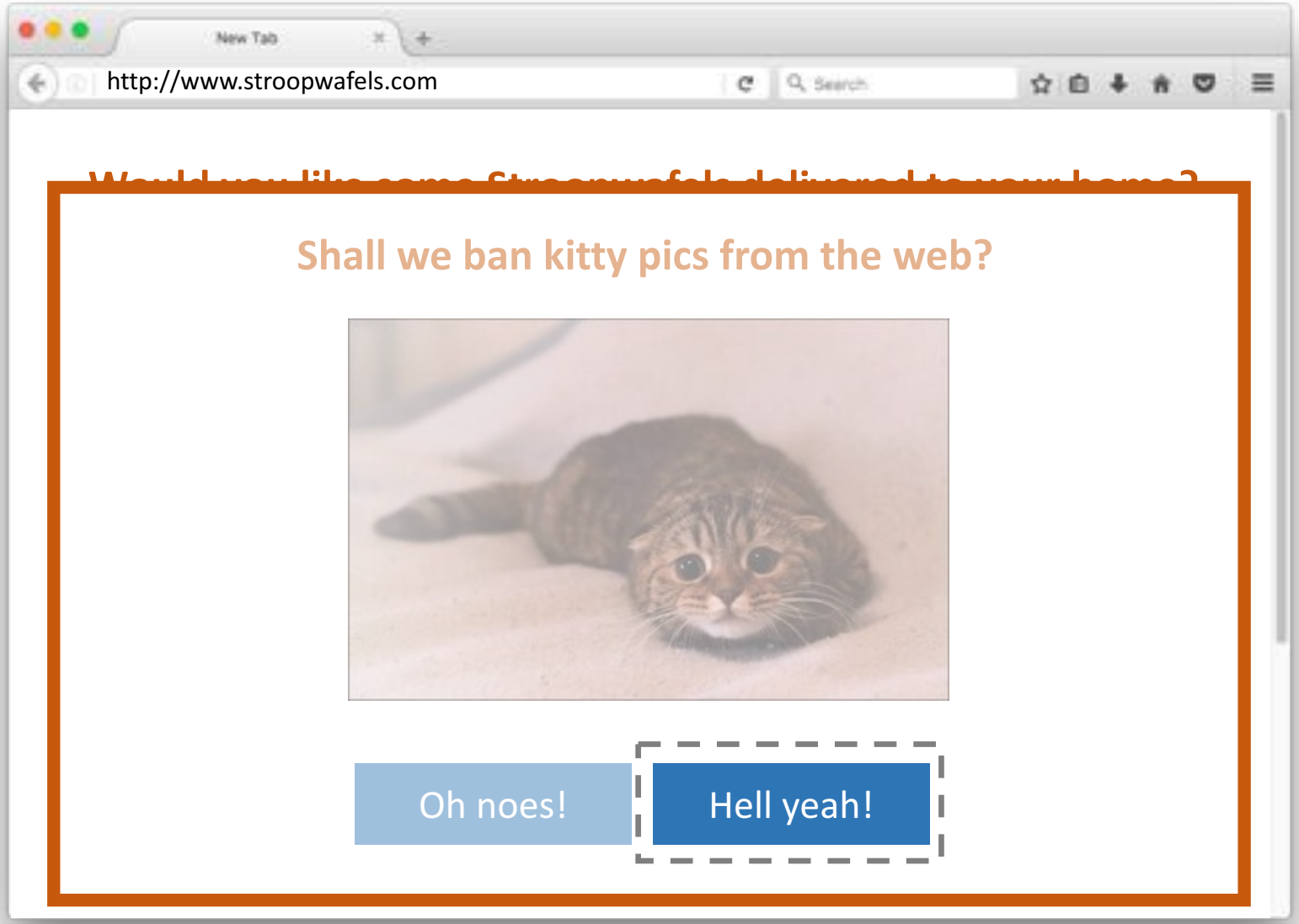
http
www.example.com
4000

http
private.example.com
4000

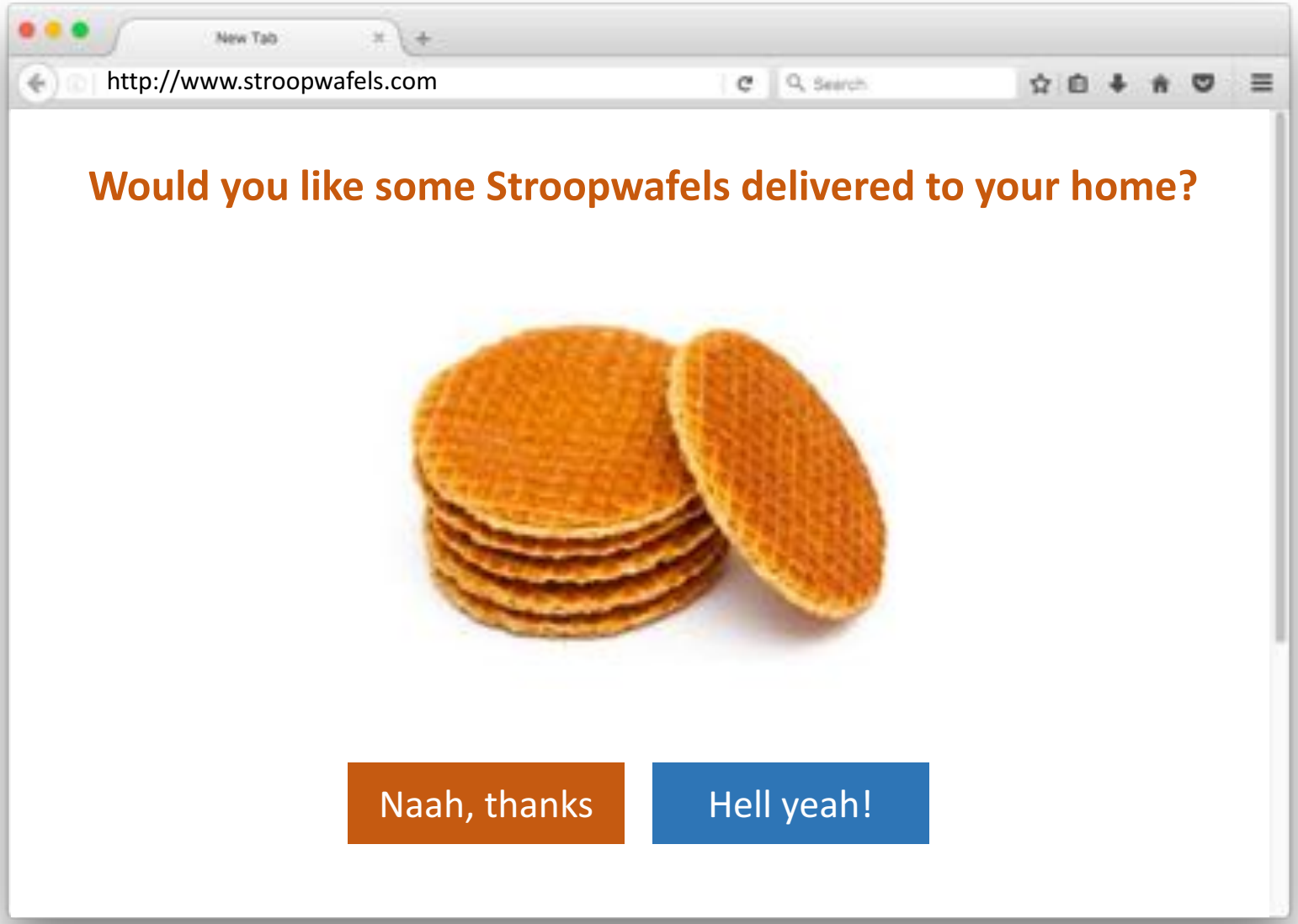
What about this?



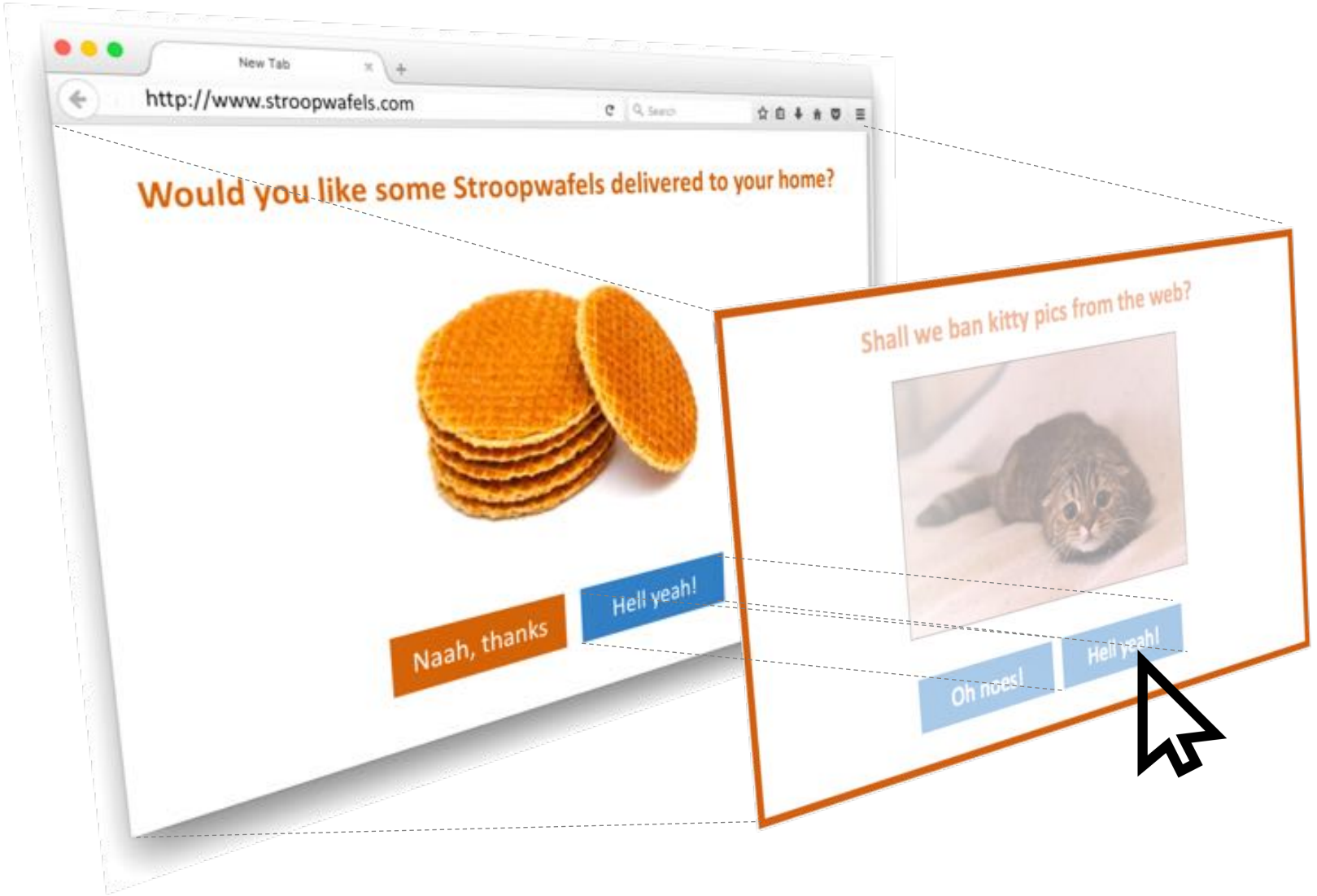
UI REDRESSING ATTACKS MISLEAD THE USER



Clickjacking is another UI redressing attack



SENDING CLICKS TO A TRANSPARENT FRAME



Preventing UI redressing attacks

- Framing is the enabler for UI redressing attacks
  - JavaScript-based framebusting is not very effective
  - Best practice is to strictly whitelist origins that are allowed to frame you

- `X-Frame-Options` header is the oldest mechanism
  - Supports `SAMEORIGIN`, `DENY` or `ALLOW-FROM` with an origin
  - `ALLOW-FROM` not supported by all browsers, so combine with `frame-ancestors`

- Content Security Policy has a `frame-ancestors` directive
  - Supports `'self'`, `'none'` or a list of allowed origins
  - Not supported by all browsers, so combine with `X-Frame-Options`

PREVENTING UI REDRESSING ATTACKS

```
X-Frame-Options: DENY
```

```
X-Frame-Options: ALLOW-FROM http://www.example.com
```

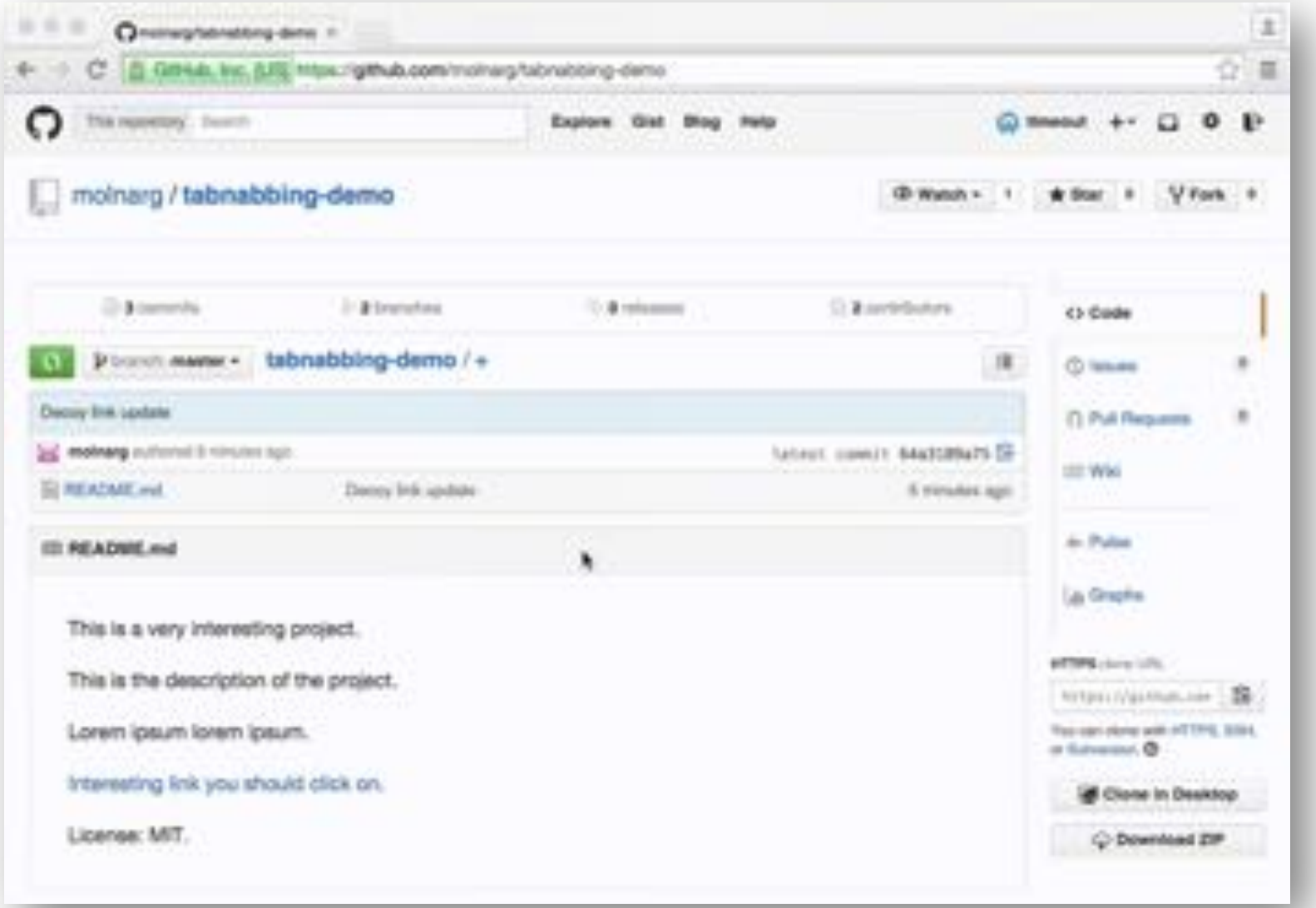
```
Content-Security-Policy: frame-ancestors http://www.example.com
```

```
Content-Security-Policy: frame-ancestors 'none'
```

BROWSER SUPPORT – X-FRAME-OPTIONS

Browser support – Content Security Policy

USING THE OPENER FOR TABNABBING



https://github.com/molnarg/tabnabbing-demo

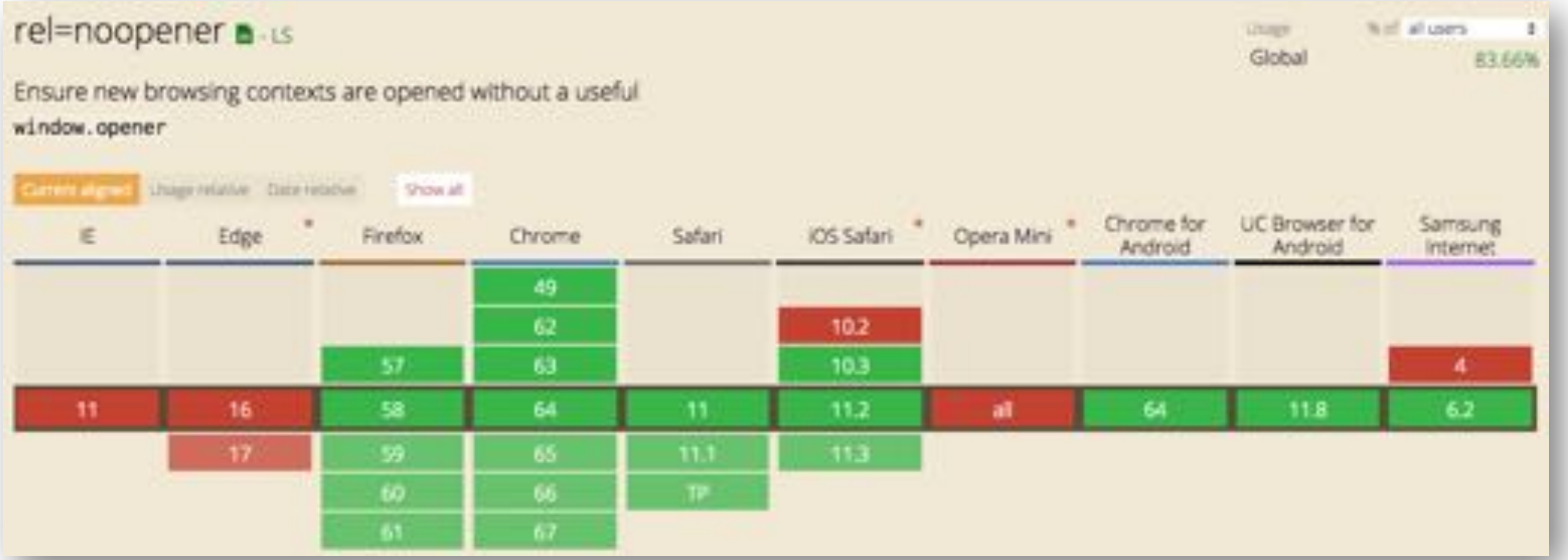
CUTTING OPENED WINDOWS LOOSE

- In most cases, there does not need to be a link back to the opener
  - The `rel` attribute on anchor tags can be set to `noopener`
  - The opener will be null, thereby preventing potential abuse

- Browser support is limited, so other options are available
  - A workaround via JavaScript, explicitly setting the opener to null before loading a page
  - The noreferrer option achieves something similar in older browsers

```
<a href="…" target="_blank" rel="noopener">…</a>
```

BROWSER SUPPORT FOR REL="NOOPENER"



Restricting framed content

- With the default security policies, framed content has a lot of freedom
  - All permissions a normal web page has
  - Possibility to navigate the top level browsing context
  - Possibility to enable full-screen mode
  - Possibility to load video or objects (Flash, Java)

- In some scenarios, you want a frame to be more restrictive
  - HTML5 introduced the **sandbox** attribute for this purpose
  - Imposes a set of restrictions on the frame, before loading the content

```
<iframe src="…" sandbox>…</iframe>
```

THE SANDBOX IS RESTRICTED BY DEFAULT

- Default set of restrictions that are applied
  - Separate, unique origin
  - No script execution
  - No form submission
  - No external navigations or popups
  - No plugin content
  - No fullscreen
  - No autoplay
  - …

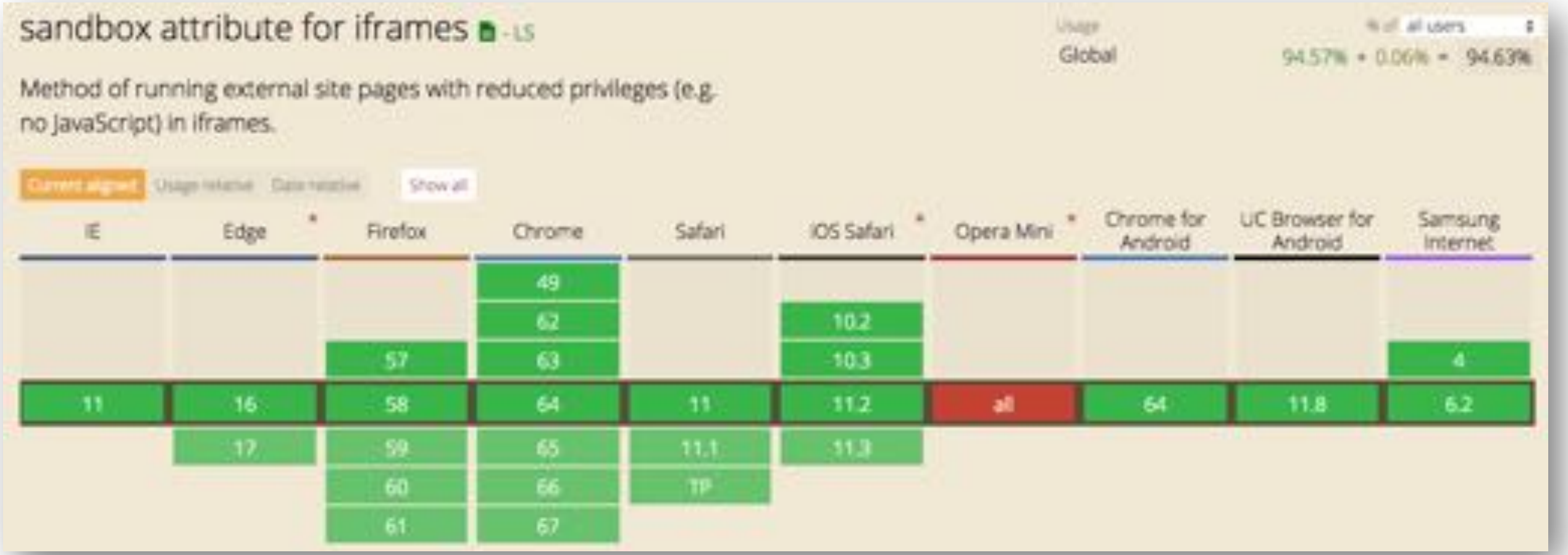
```
<iframe src="…" sandbox>…</iframe>
```

Relaxing the sandbox

- Restrictions can be lifted by adding specific keywords
  - E.g. allow-scripts, allow-same-origin, …
- Some restrictions cannot be lifted
  - Plugin content cannot be re-enabled
  - Navigating arbitrary contexts is not allowed (only top-level or auxiliary)
- **Enabling allow-scripts together with allow-same-origin is dangerous**
  - Allows the sandboxed script to break out of the sandbox

```
<iframe src="…" sandbox="allow-scripts">…</iframe>
```

ALL BROWSERS PROVIDE A SANDBOXED IFRAME



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

COMBINING SANDBOX WITH SRCDOC

- Sandboxing is really powerful when combined with **`srcdoc`**
  - Lightweight mechanism to load content in an isolated environment
  - Directly specify the HTML in the attribute, without requiring a page load first
  - Use the **`sandbox`** attribute to leverage the SOP and apply additional restrictions

- The src attribute can be used as a fallback mechanism
  - Supporting browsers will use **`srcdoc`** and ignore **`src`**
  - Older browsers ignore **`srcdoc`** and use **`src`**

```
<iframe src="…" srcdoc="<p>…</p>" sandbox>…</iframe>
```

COMMUNICATION BETWEEN BROWSING CONTEXTS

- Until HTML5, there was no designed communication channel
  - Hacky workarounds leveraged the URI fragment to send messages
  - Today, we have the Web Messaging API

```
frame.contentWindow.postMessage("Moar Wafels", "http://www.example.com");
```

```
window.addEventListener("message", function(e) {
    if(e.origin === "http://wafels.example.com") {
        console.log("Incoming message: " + e.data);
    }
}
```

COMMUNICATING WITH A SANDBOXED CONTEXT

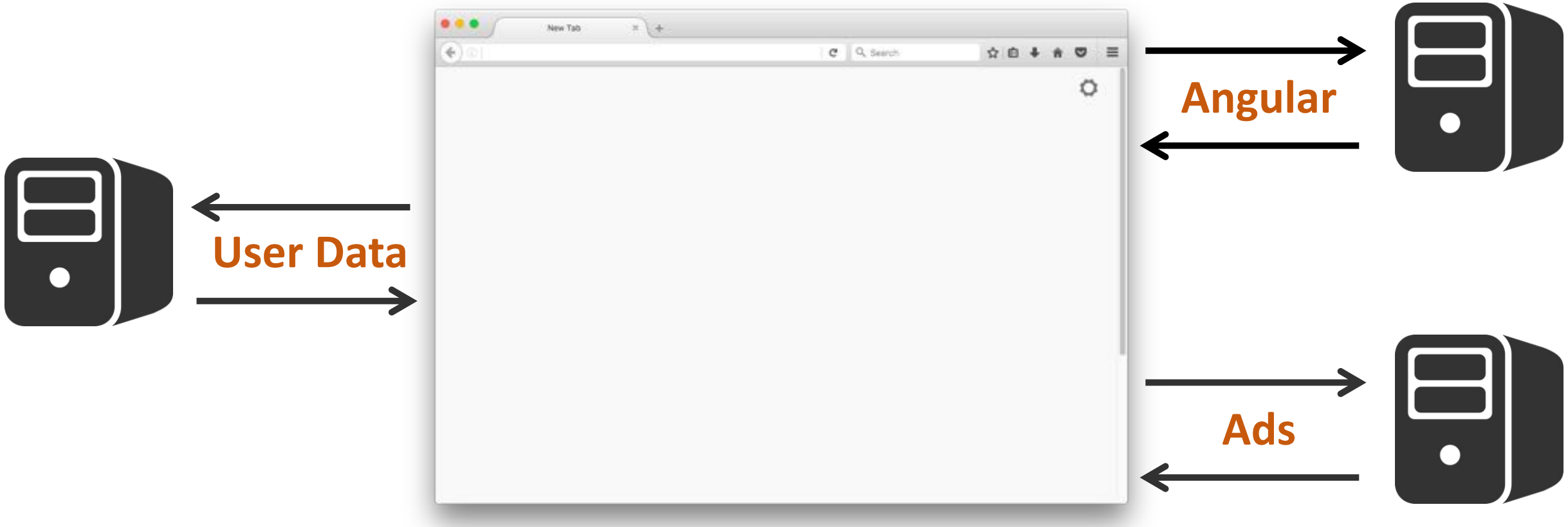
- A sandboxed content has a unique origin
  - This is canonicalized as `null`, which is not a valid origin
  - For Web Messaging, this means using the wildard `*`

```
frame.contentWindow.postMessage("Moar Wafels", "*");
```

```
window.addEventListener("message", function(e) {
    if(e.origin === "http://wafels.example.com") {
            console.log("Incoming message: " + e.data);
    }
}
```

Script Contexts

SCRIPTS CAN COME FROM ANYWHERE



Script contexts and browsing contexts

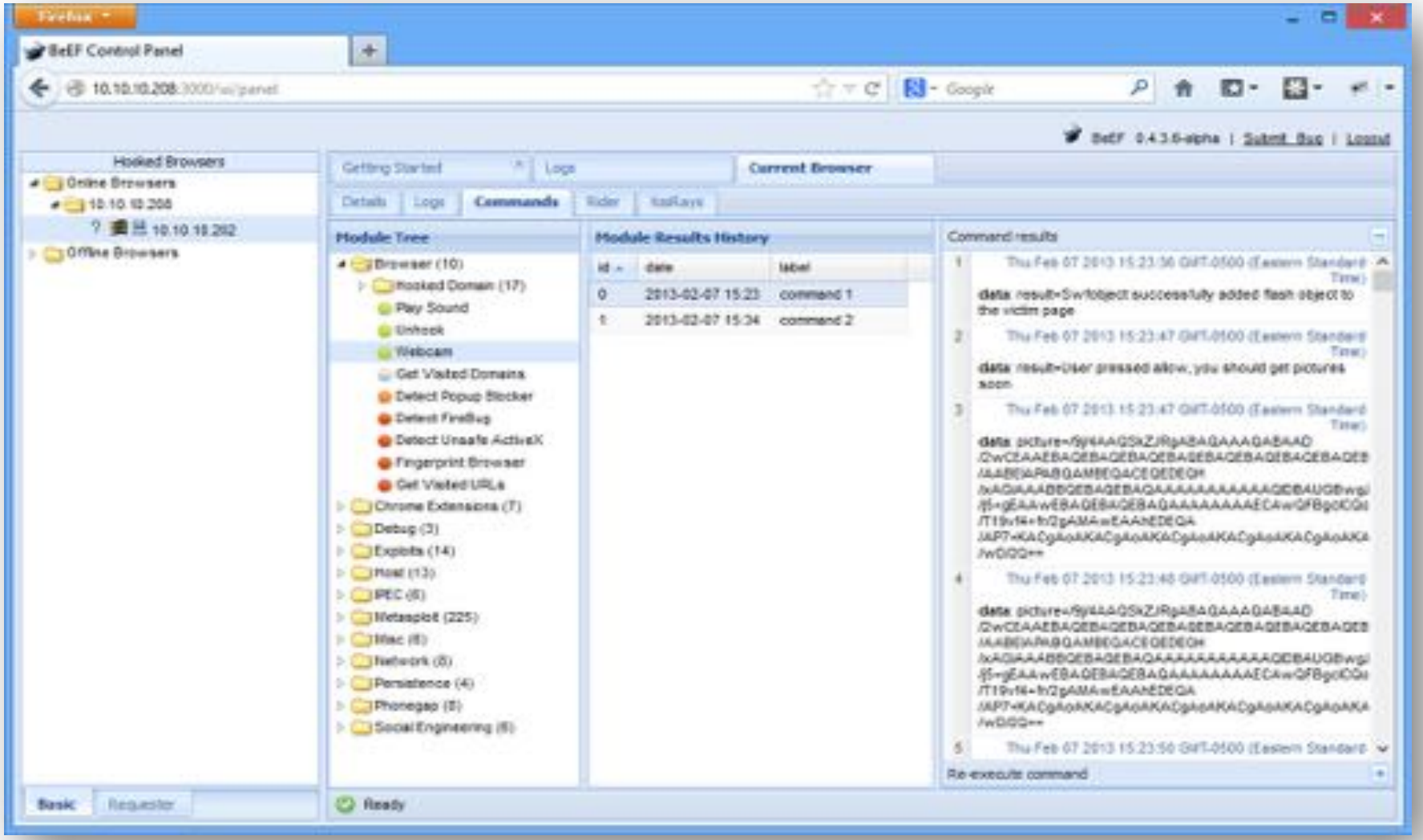
- Unlike documents, scripts are not loaded in a separate context
  - Each browsing context only has one script context
  - All scripts in the document run within this one context
  - The browsing context has one shared scope and namespace

- The lack of code isolation has resulted in a few serious security problems
  - User injected script runs within the document's context (Cross-Site Scripting)
  - Including an external library requires full trust in the third-party provider
  - It is common practice to embed third-party components without any isolation

Cross-Site Scripting (XSS)

- In an XSS attack, malicious content is injected into your application's pages
  - In the "original" XSS attacks, an attacker injected JavaScript code
  - Today, injected content can be JavaScript, CSS, HTML, SVG, …

THE TRUE POWER BEHIND XSS



BeEF
THE BROWSER EXPLOITATION FRAMEWORK PROJECT

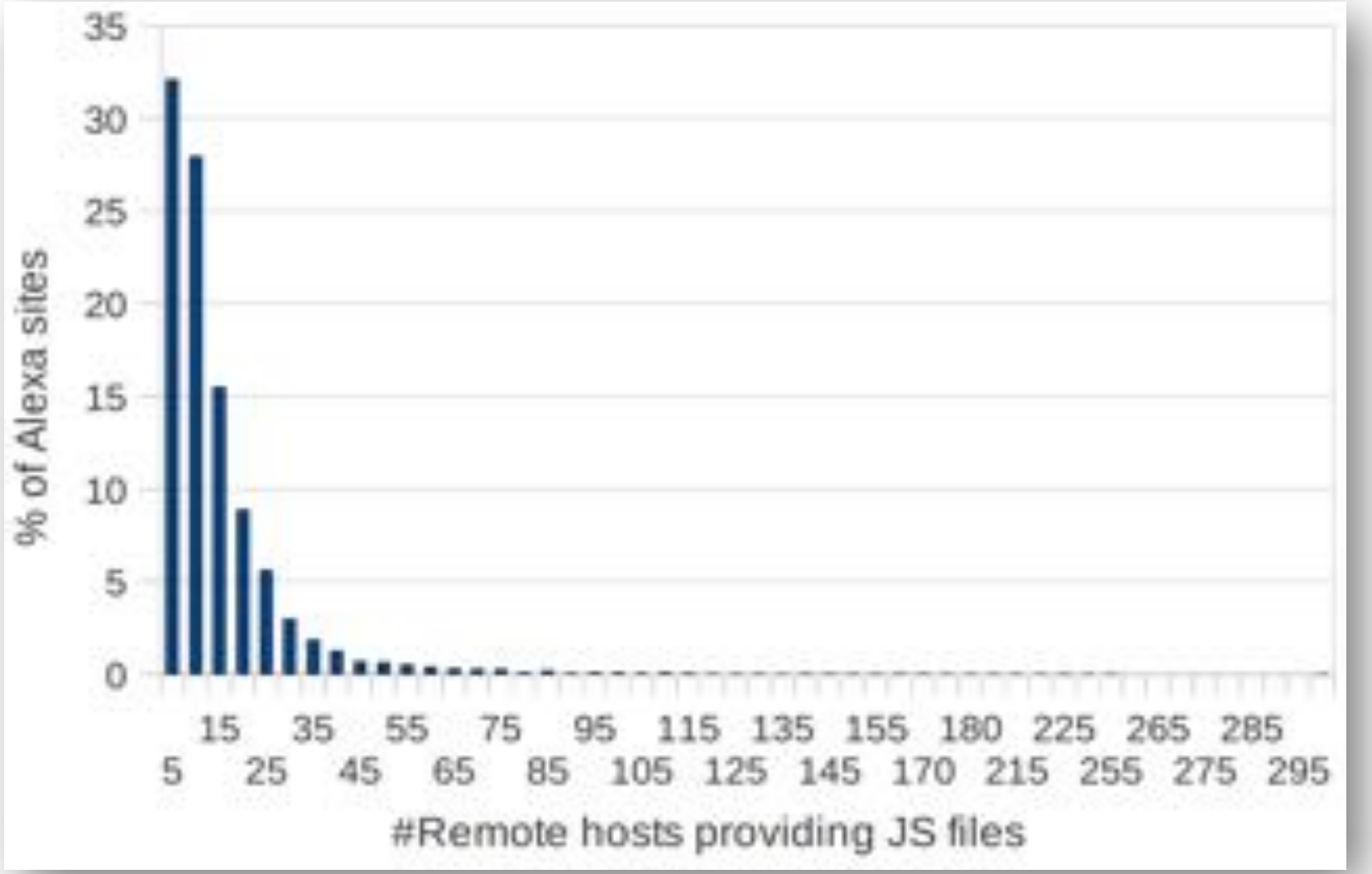
http://colesec.inventedtheinternet.com/beef-the-browser-exploitation-framework-project/

YOU ARE WHAT YOU INCLUDE …

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

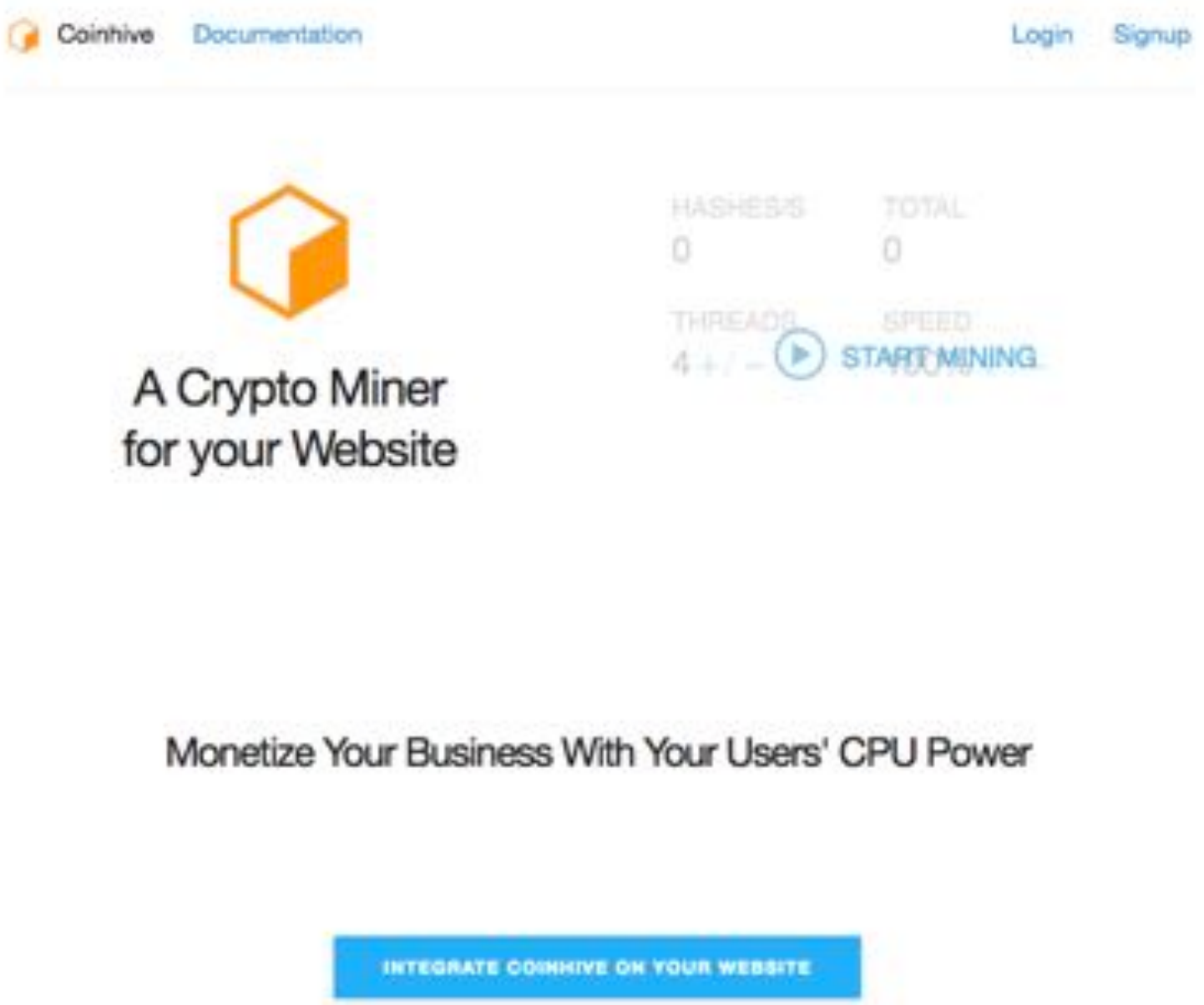
https://seclab.cs.ucsb.edu/media/uploads/papers/jsinclusions.pdf

You are what you include …

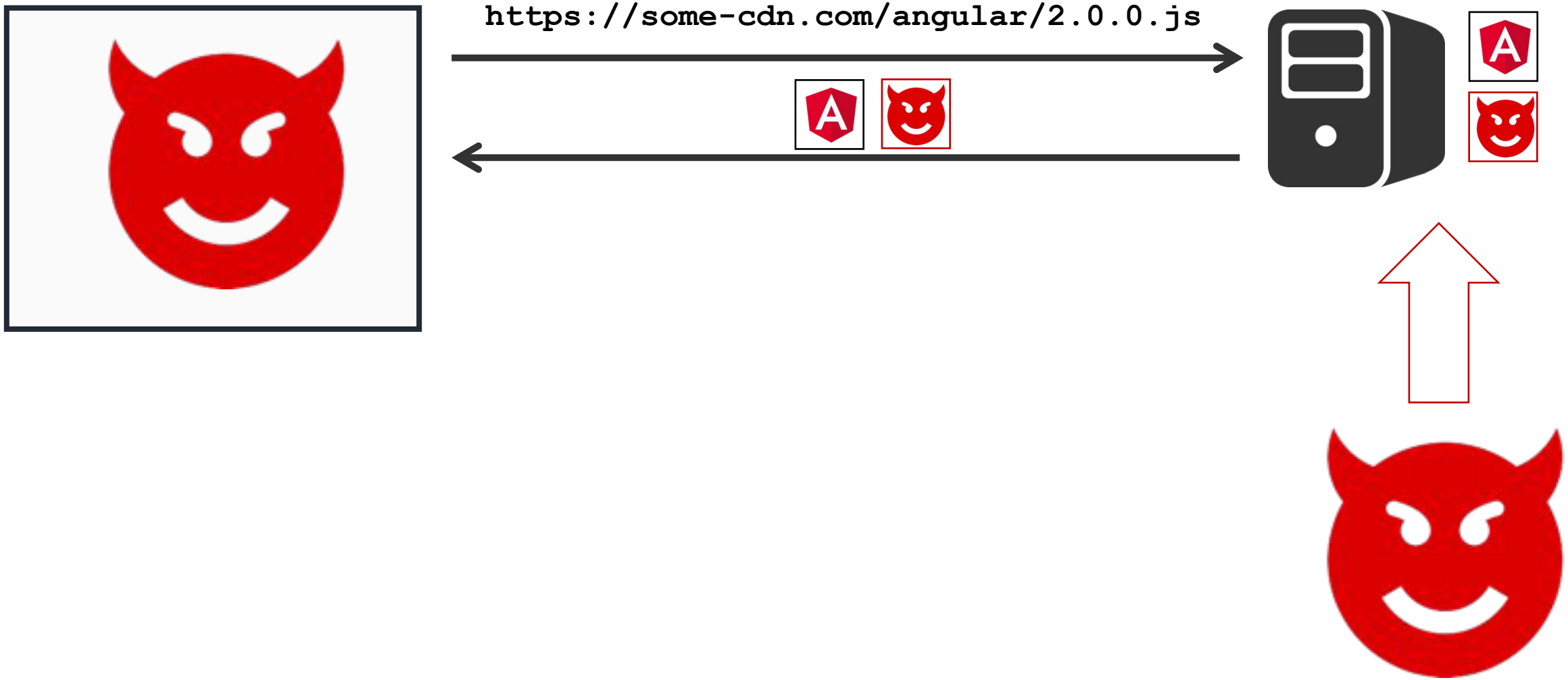


https://seclab.cs.ucsb.edu/media/uploads/papers/jsinclusions.pdf


Google Analytics
Anywhere. Anytime.
Everywhere


Coinhive
Documentation
Login
Signup
A Crypto Miner
for your Website
HASHES/S
TOTAL
0
0
THREADS
SPEED
START MINING
Monetize Your Business With Your Users' CPU Power
INTEGRATE COINHIVE ON YOUR WEBSITE

When you load a script, all you have is a name ...



Massive denial-of-service attack on GitHub tied to Chinese government

Reports: Millions of innocent Internet users conscripted into Chinese DDoS army.

Now researchers have unearthed additional evidence implicating China that goes beyond motive. Specifically, the computers hammering GitHub servers are all running a piece of malicious code that surreptitiously makes them soldiers in a massive DDoS army. The JavaScript gets silently injected into the traffic of sites that use an analytics service that China-based search engine Baidu makes available so website operators can track visitor statistics. About one percent of people visiting such sites don't receive the true Baidu analytics JavaScript but instead get code that forces their browser to constantly reload the two targeted GitHub pages.

https://arstechnica.com/security/2015/03/massive-denial-of-service-attack-on-github-tied-to-chinese-government/

Know what you load with Subresource Integrity

```
<script src="https://…2.0.0.js"
  integrity="sha384-814a3…Yzi=">
</script>
```

https://…/2.0.0.js

Verify checksum

```
Failed to find a valid digest in the 'integrity' attribute for resource   sri.html:1
'https://cdnjs.cloudflare.com/ajax/libs/angular.js/2.0.0-beta.17/angular2.js' with
computed SHA-256 integrity 'pQ+zWKiHP91iLkd/wohYUH/XvvabBTRKl9UjoIPFh5U='. The
resource has been blocked.
```

DATA LEAKAGE THROUGH SRI

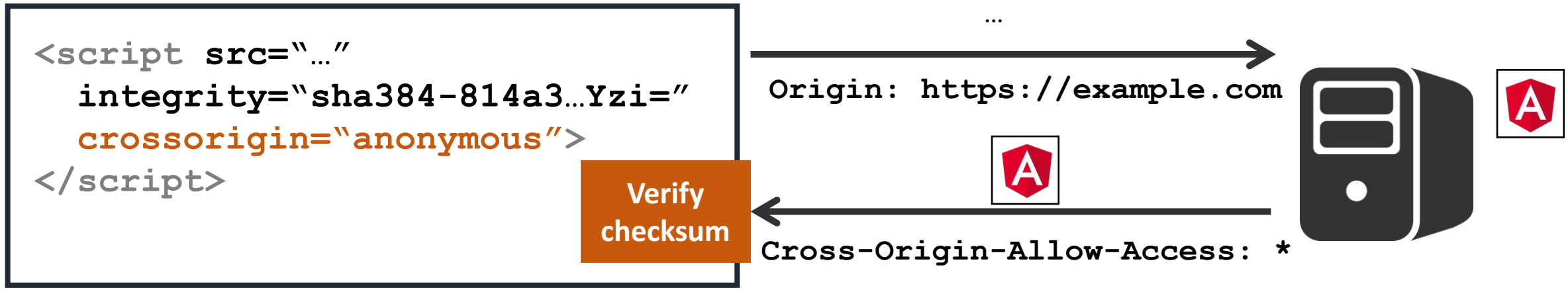
```
<script src="https://…/api/accountbalance.js"
  integrity="sha256-…="
  crossorigin="use-credentials"></script>
```

{"balance": 1234.00}	dPdFnnWdXY6eHXiK+3O/OSi3OeLFHlLch1qZ3iqD3MGNXck+Oz4LETv8lnsoNyFI
	⊗ Failed to find a valid digest in the 'integrity' attribute for resource
{"balance": 1235.00}	RasWnvVTFAiT+6NeqIJFRDDDSklMaljV0FxUQysJqUB65TGm/lFqKJkrGif2wzYj
	⊗ Failed to find a valid digest in the 'integrity' attribute for resource
{"balance": 1236.00}	uSCKm1yloPZ7VexjyLQ+sUvakZKycl3CsblGH/9XpGV09ymyf1nKAzU5tXTFH5oi
	⊗ Failed to find a valid digest in the 'integrity' attribute for resource
{"balance": 1237.00}	4SI2gcfIFhX2NRE5KPbeXR87PaiCSAan6PL2mxKWndBp8wvE2Dfcn7HenpNXD0oJ

On the web, it's never that simple …

- SRI allows an attacker to determine the existence of a predetermined file
  - If no error is generated, the checksum matches and the file exists

- To avoid this privacy leak on legacy servers, CORS must be used
  - The server needs to opt-in to use SRI by sending a CORS response header
  - Can either be anonymous (no cookies) or authenticated (with cookies)



On a positive note, many CDNs make SRI really easy


```
<script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/2.0.0-beta.17/angular2.js" integrity="
sha256-pQ+zWKiHP91iLkd/wohYUH/XvvabBTRKl9UjoIPFh5U=" crossorigin="anonymous"></script>
```

https://cdnjs.com/libraries/angular.js/

BUT DOING IT YOURSELF IS NOT VERY HARD

https://www.srihash.org/

WIDESPREAD BROWSER SUPPORT IS COMING

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

Leveraging browsing contexts for privilege separation

- Different browsing contexts can have different privileges
  - All contexts within the same origin will have the same privileges (permissions, data, …)
- Privilege separation is possible, but requires some effort
  - Works well for standalone components
  - Difficult for cross-cutting libraries, such as JS frameworks, analytics code, …
- Privilege separation in practice
  - Loading a document from a different origin leverages the SOP
  - Loading a document in a sandboxed frame creates a unique origin
  - Communication can be enabled with the Web Messaging API

Privilege separation at Dropbox

```
function startSupportChat() {
    SnapEngage.setWidgetId(SUPPORT_ID);
    SnapEngage.setUserEmail(chatData.Email, true)
    SnapEngage.startChat("How can we help you today?")
}
```

```
DropboxSnapEngage.startSupportChat = function() {
    this.chatRequested = true;
    DropboxSnapEngage.showSnapEngageIframe();
    return DropboxSnapEngage.sendMessage({
        'message_type': 'startSupportChat',
        'chatData': this.chatData
    });
};

DropboxSnapEngage.sendMessage = function(data) {
    var content_window;
    content_window = DropboxSnapEngage.getSnapEngageIframe().contentWindow;
    return content_window.postMessage(data, this.SNAPENGAGE_IFRAME_ORIGIN);
};
```

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

The goal of Content Security Policy (CSP)

- CSP is intended as a defense-in-depth mechanism against injection attacks
  - Gives developers a way to lock down their application in various ways
  - Constrains an attacker in case of an injection vulnerability in the application
  - ***CSP is not a replacement for traditional XSS mitigation techniques***

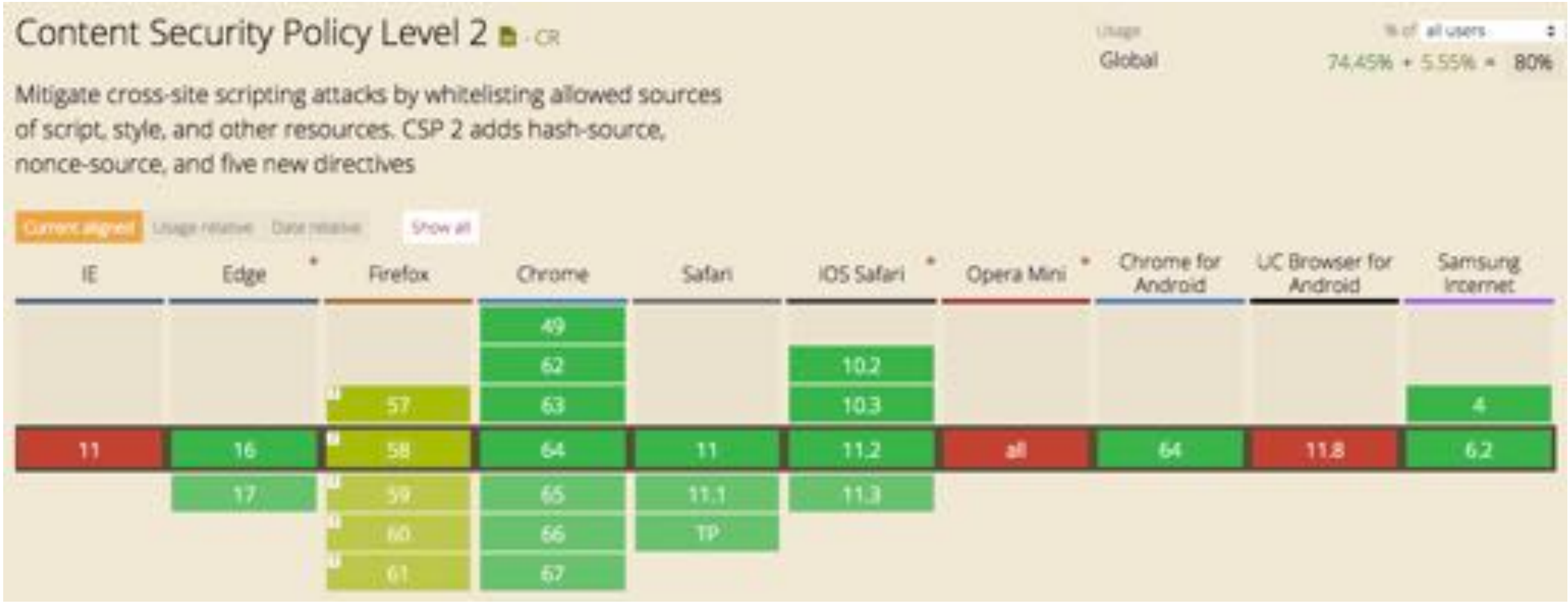
- CSP places two kinds of restrictions on a page
  - It disables "dangerous features" (e.g. inline scripts, inline styles and the use of eval)
  - It only loads resources that are explicitly whitelisted, and blocks everything else

- CSP is an extensive security policy, with a wide variety of features
  - We will focus on its capabilities to restrict XSS attacks first

CSP CAN ALSO RESTRICT OTHER TYPES OF CONTENT

- Injection attacks do not necessarily depend on JavaScript
  - CSS injection can allow for the extraction of information
  - HTML injection can modify the UI, tricking the user into performing certain actions
- CSP has plenty of directives to constrain behavior in the context
  - Directives to control included content (styles, images, fonts, frames, …)
  - Directives to control outgoing requests (XHR, form submissions, ...)
  - Directives to define a sandbox on the current resource
- Additionally, other security features have been added to CSP as well
  - The mechanism to upgrade insecure requests and to block mixed content
  - A replacement mechanism for the X-FRAME-OPTIONS header

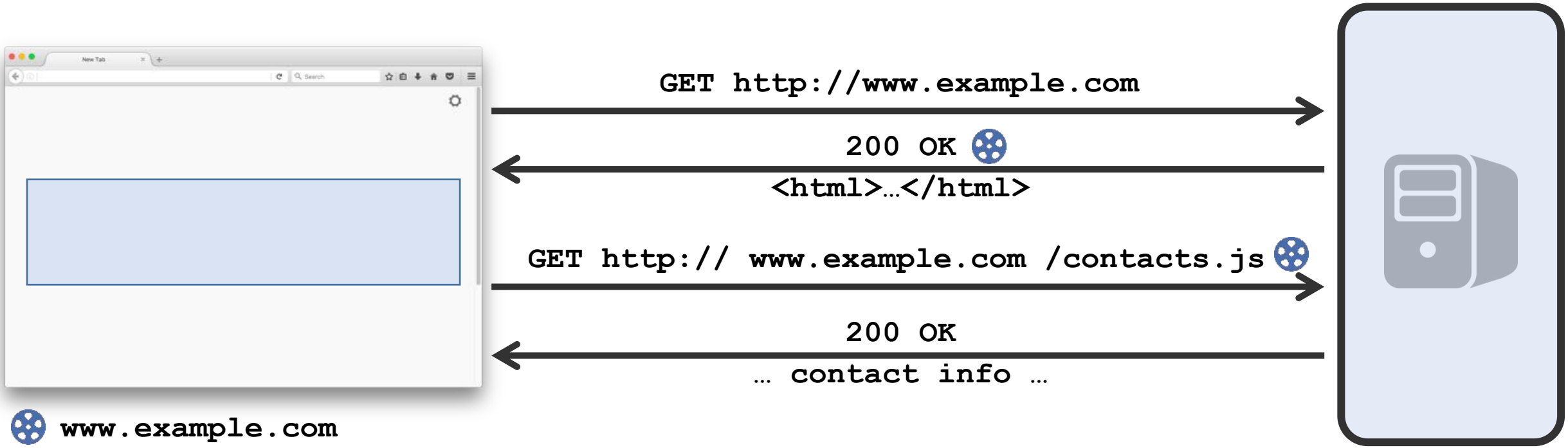
Browser support – Content Security Policy



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

SESSIONS, COOKIES AND TOKENS

COOKIE-BASED SESSION MANAGEMENT



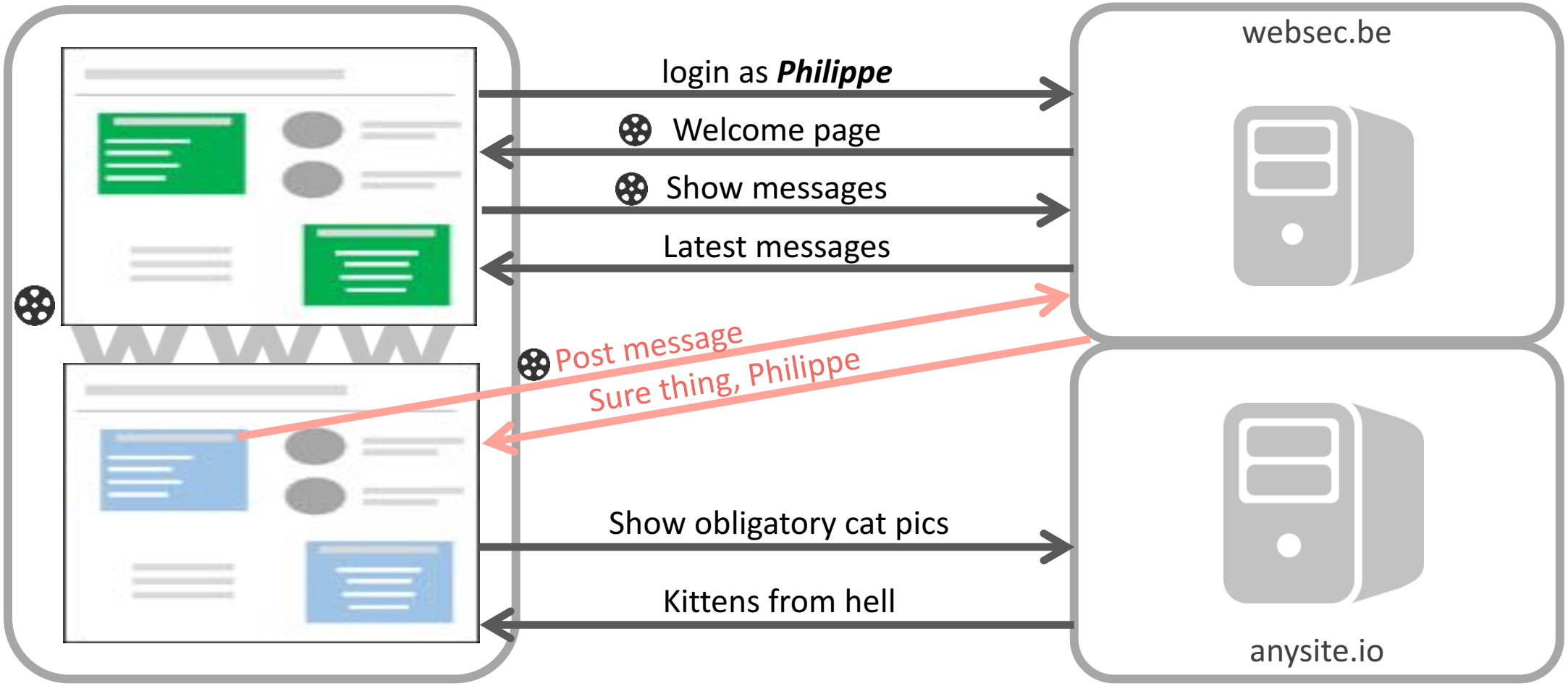
The security properties of cookies

- Cookies are associated with a domain, not with an origin
  - Cookies are shared over HTTP and HTTPS
  - Cookies can be read and set by a header, and from JavaScript
- These properties are suboptimal, and cause a lot of problems
  - Stealing cookies through eavesdropping
  - Session hijacking / session fixation through JavaScript
- Cookie flags aim to patch cookie behavior to make it more secure
  - The `Secure` flag marks a cookie for use over HTTPS only
  - The `HttpOnly` flag makes a cookie inaccessible from JavaScript

Cookie prefixes make it even more complicated

- The recently proposed cookie-prefix spec tries to restrict cookie behavior
  - Cookie names can be prefixed with an attribute, enforcing strict behavior
- The ***__Secure-*** prefix restricts a cookie to secure connections only
  - It cannot be set over an insecure connection
  - It cannot be set if the ***Secure*** flag is missing
- The ***__Host-*** prefix restricts a cookie to a specific host
  - It will only be sent to a host, never to a domain
  - It must be set for the root path (/) and with the ***Secure*** flag
- Enforcement depends on browser behavior
  - Currently supported in all modern browsers (Chrome, Firefox, Opera, Edge, Safari)

THE UNDERESTIMATED THREAT OF CSRF



The essence of CSRF

- CSRF exists because the browser handles cookies very liberally
  - They are automatically attached to any outgoing request
  - By default, there's no mechanism to indicate the intent of a request
- Many applications are unaware that any context can send requests
  - The session cookies will be attached automatically by the browser
  - Defending against CSRF requires explicit action by the developer
- Because of its subtle nature, CSRF is a common vulnerability
  - Illustrated by cases at Google, Facebook, eBay, …
  - Ranked #8 on OWASP top 10 (2013)

HIJACKING ACCOUNTS USING CSRF

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

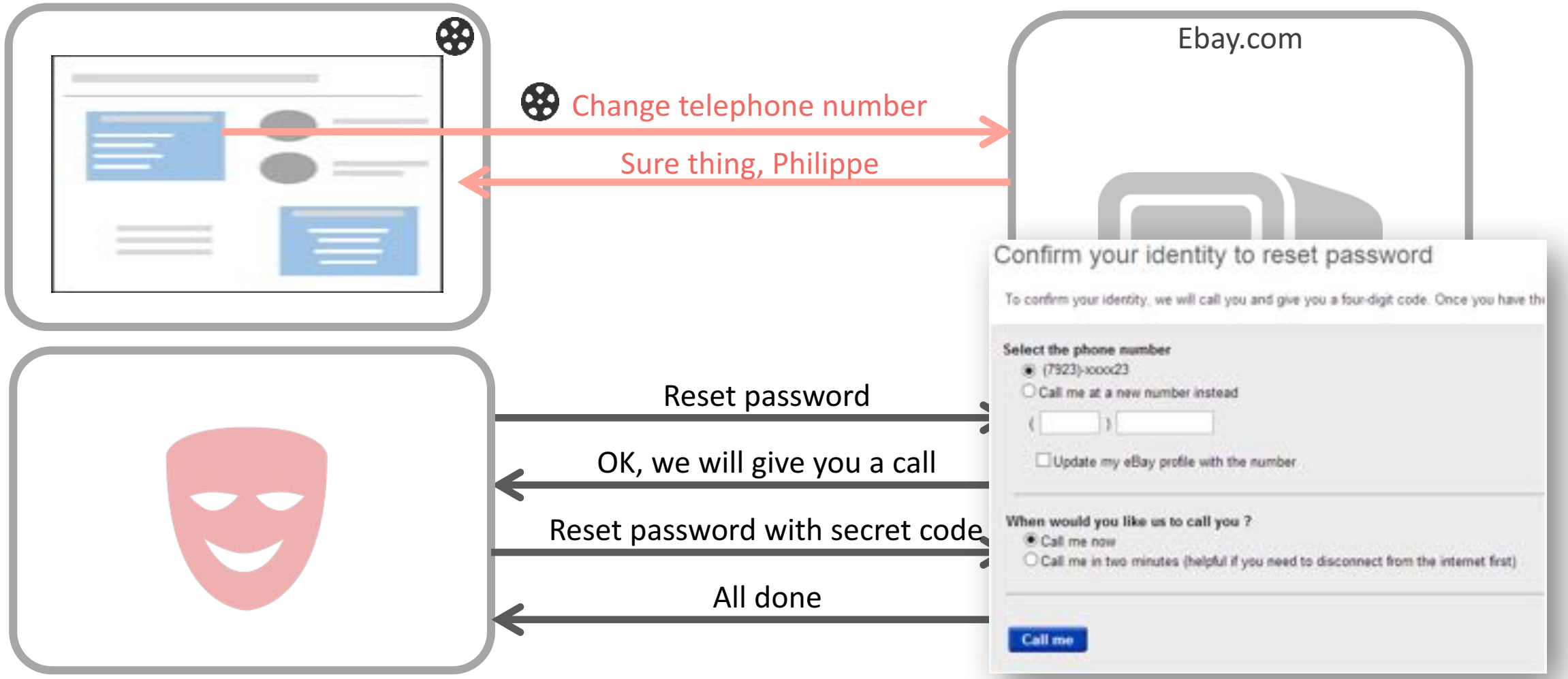
The issue has been reported to eBay, but it's still unfixed

Sep 16, 2013 11:10 GMT · By Eduard Kovacs · Share:

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.

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

HIJACKING ACCOUNTS USING CSRF

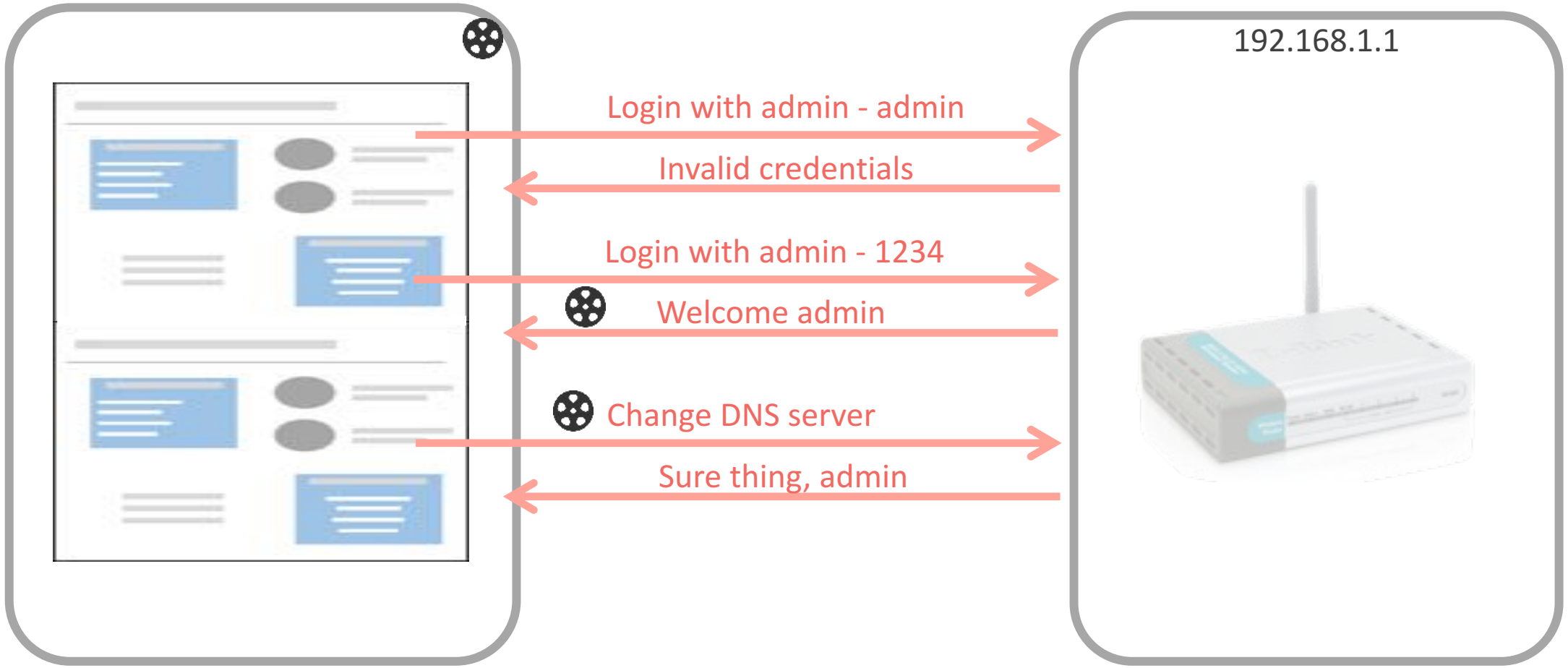


http://news.softpedia.com/news/CSRF-Vulnerability-in-eBay-Allows-Hackers-to-Hijack-User-Accounts-Video-383316.shtml

TAKING CONTROL OF YOUR HOME NETWORK WITH CSRF

http://support.dlink.com/CMS_FTP/CMS_DAF/Product/Pictures/DI-524/DI-524_fornt20131003114340.png

TAKING CONTROL OF YOUR HOME NETWORK WITH CSRF



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

TAKING CONTROL OF YOUR HOME NETWORK WITH CSRF

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

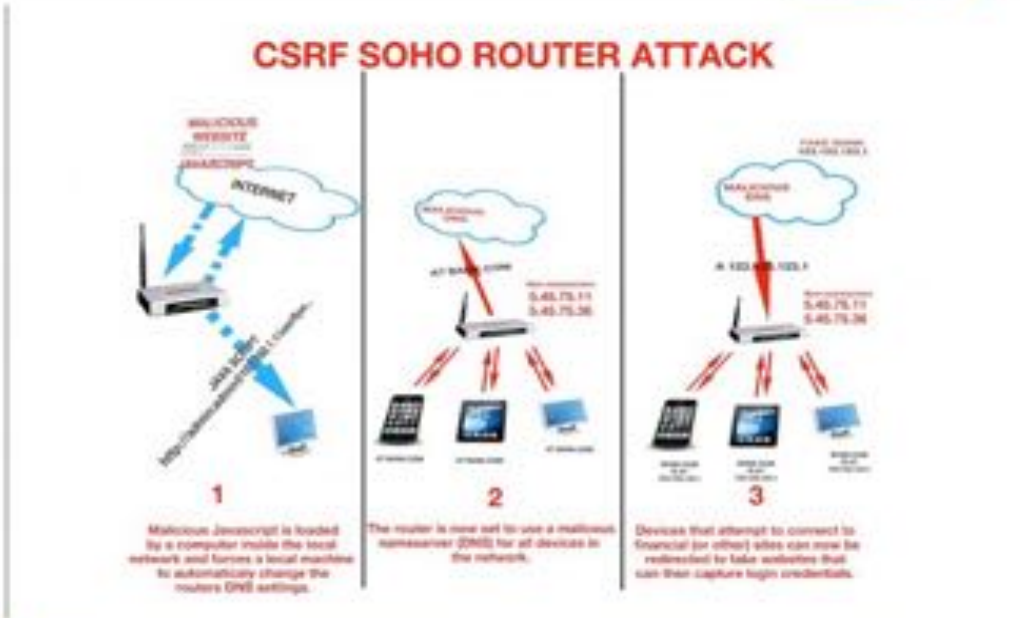
Pharming attacks are generally network-based intrusions where the ultimate goal is to redirect a victim's web traffic to a hacker-controlled webserver, generally through a malicious modification of DNS settings.

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



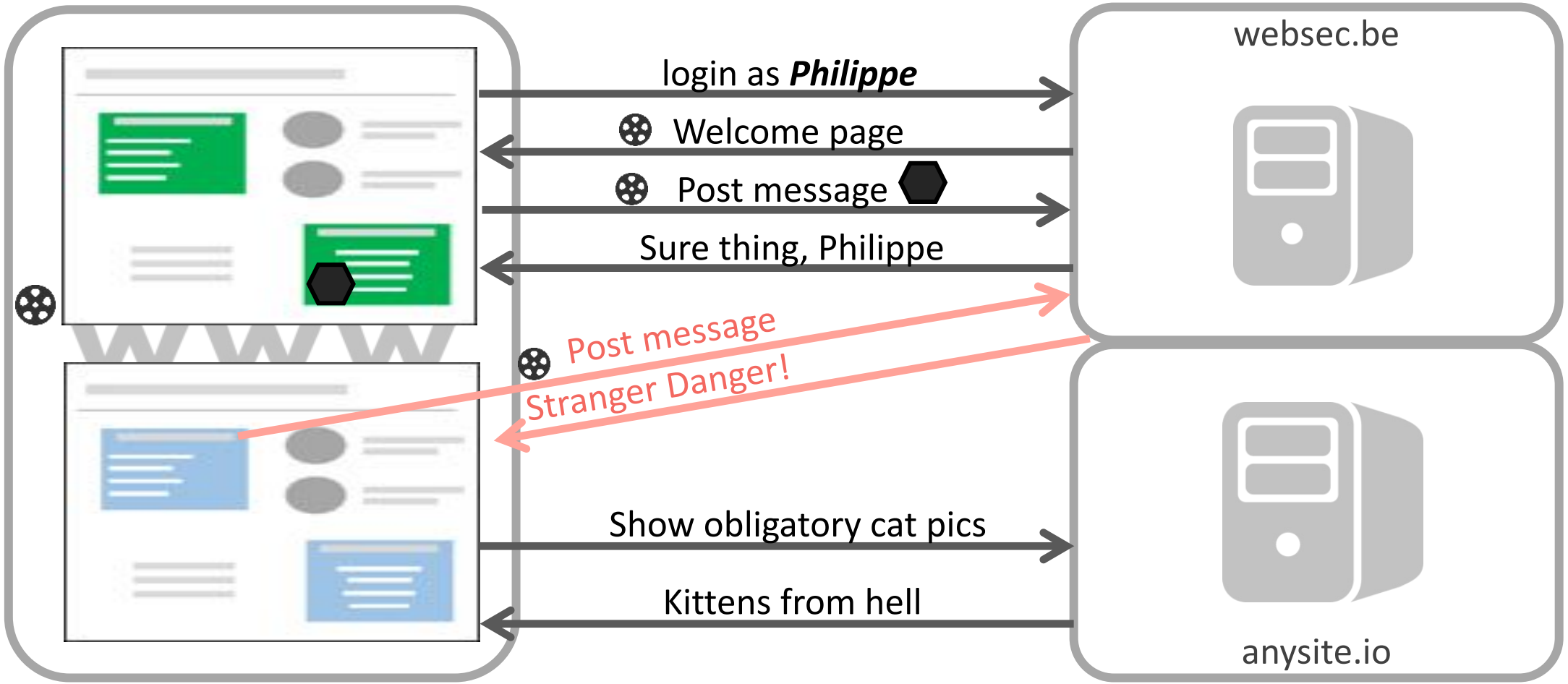
Enlarge / Three phases of an attack that changes a router's DNS settings by exploiting a cross-site request vulnerability in the device's Web interface.

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.

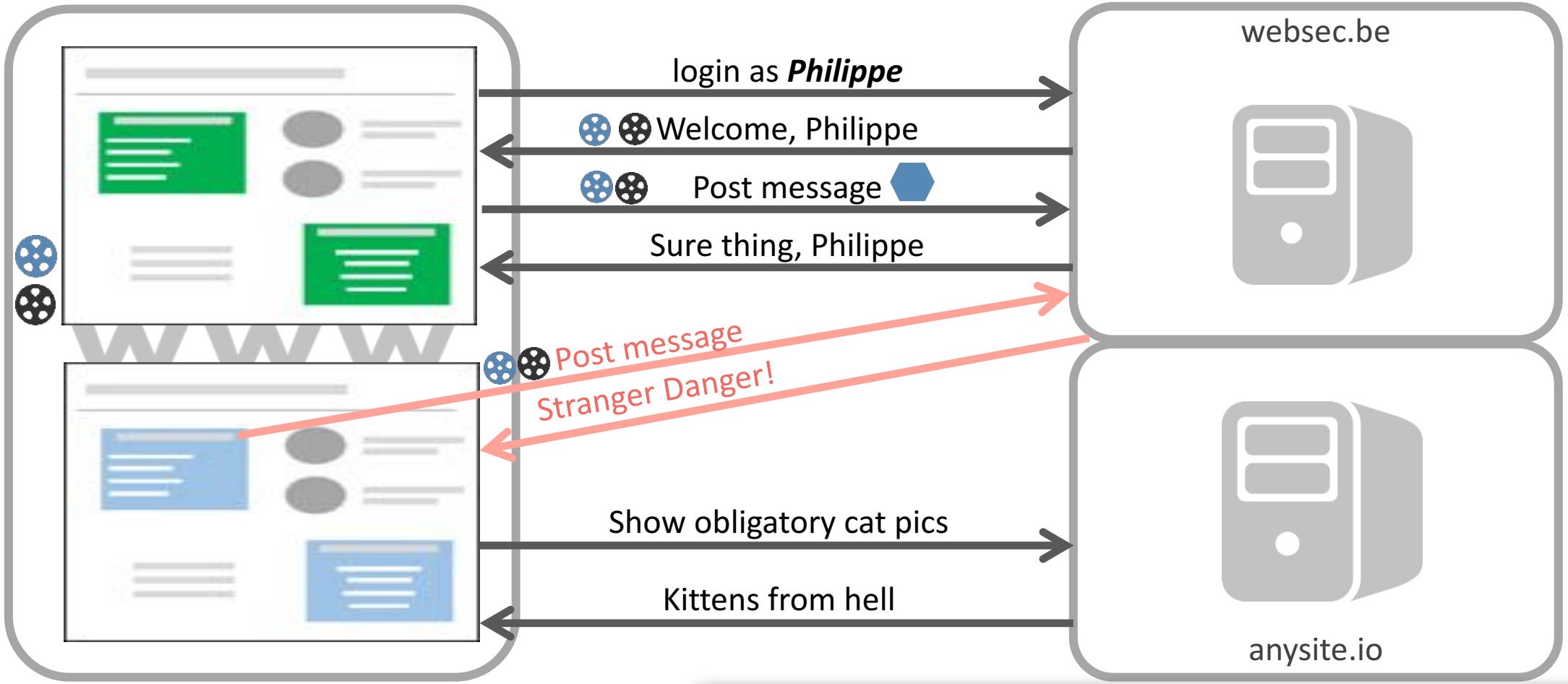
http://arstechnica.com/security/2014/03/hackers-hijack-300000-plus-wireless-routers-make-malicious-changes/
https://threatpost.com/pharming-attack-targets-home-router-dns-settings/111326

CSRF Defense 1: hidden form tokens




```
<input type="hidden" name="csrftoken" value"1234abc" />
```

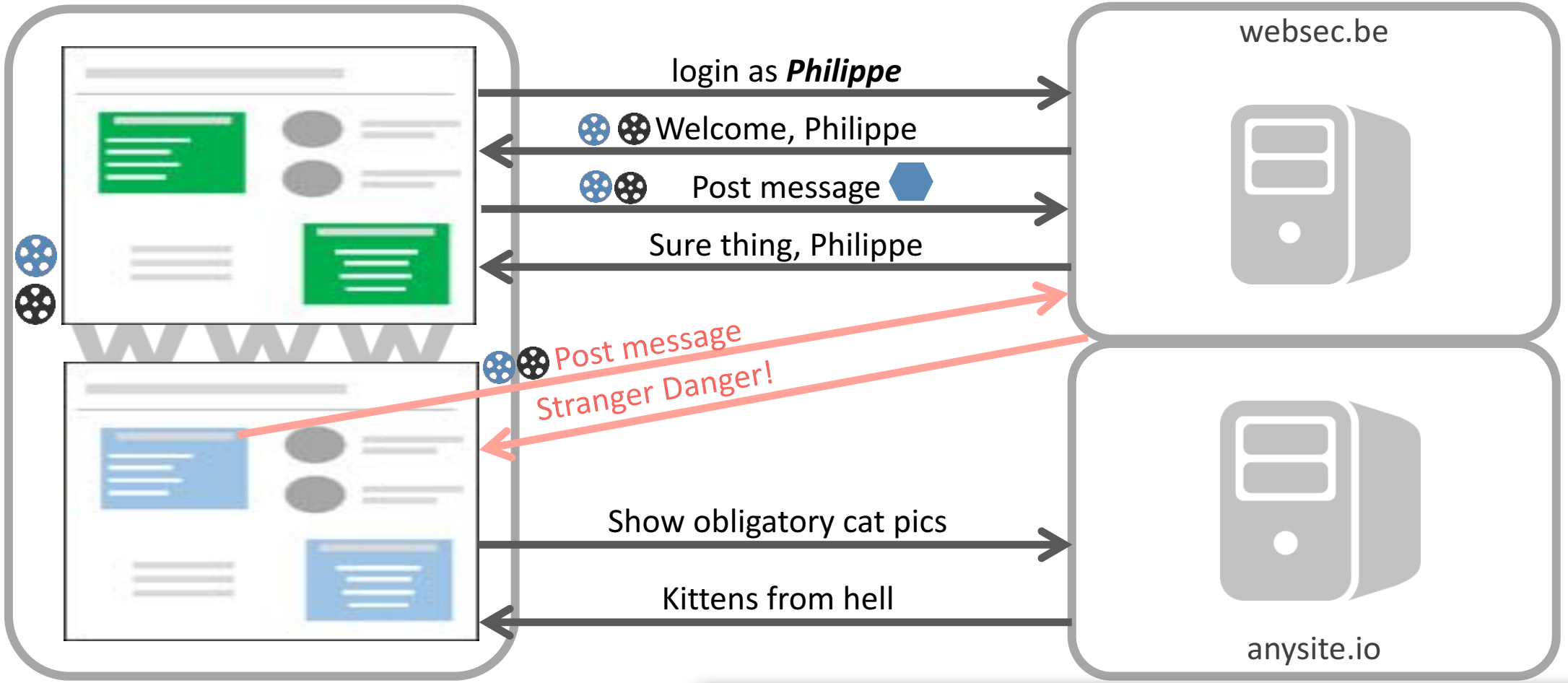
CSRF Defense 2: transparent tokens



Cookie value is copied to a header by JavaScript code

```
POST …
Cookie: SID=123, XSRF-TOKEN=abc
X-XSRF-TOKEN: abc
```

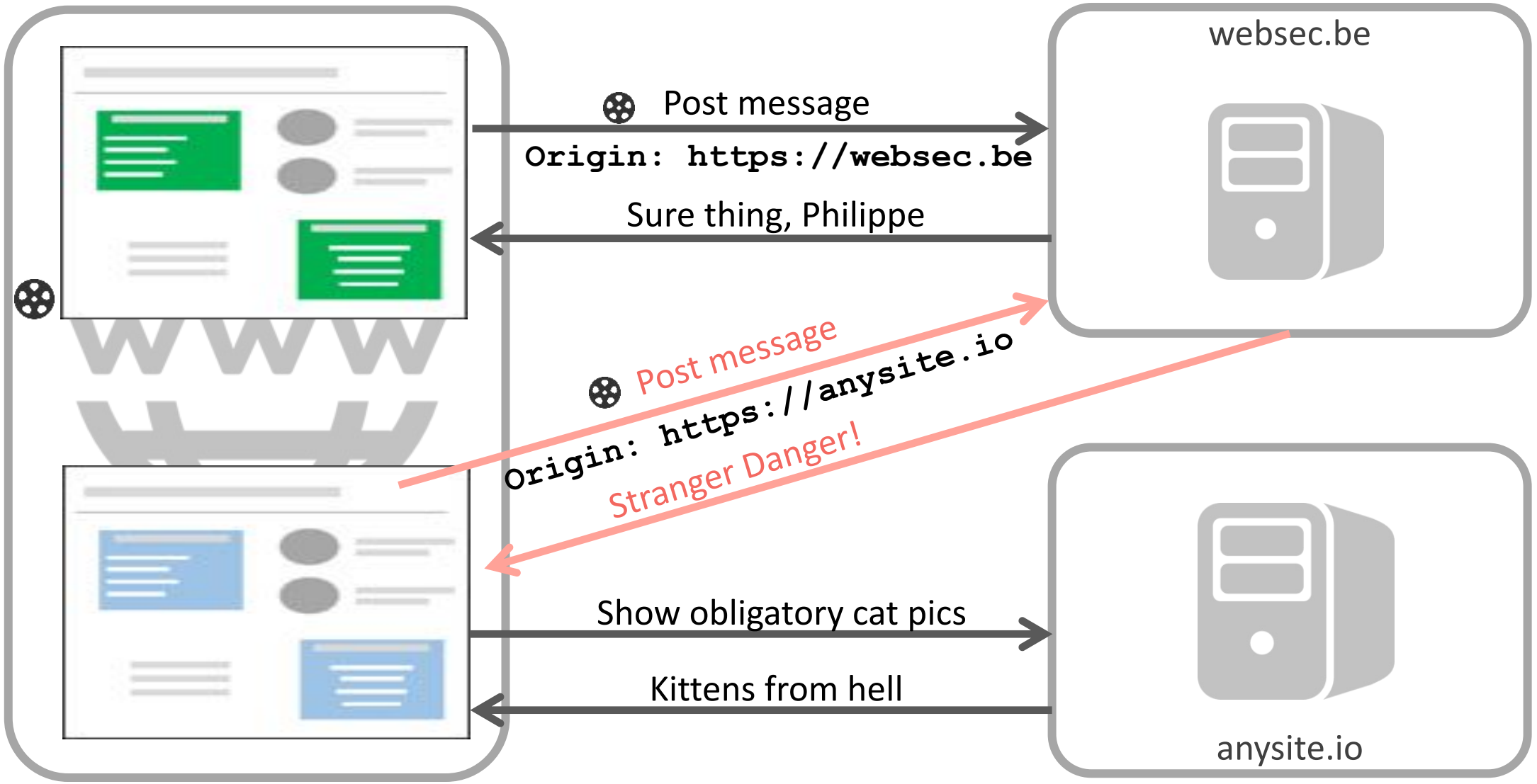
CSRF Defense 2bis: transparent tokens



Cookie value is copied to a hidden form field by JS

```
POST …
Cookie: SID=123, XSRF-TOKEN=abc
…&xsrftoken=abc
```

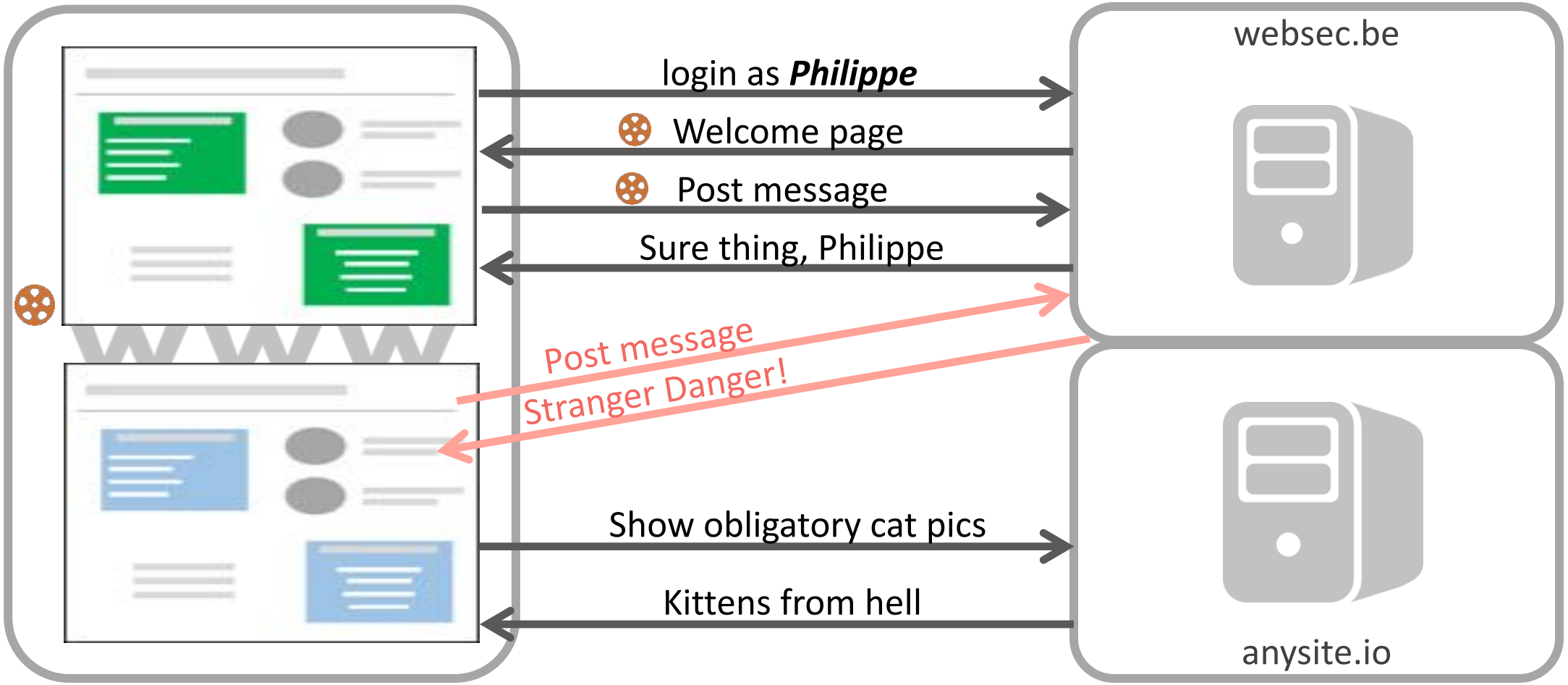
CSRF Defense 3: checking the Origin header



Reliability issues with the Origin header

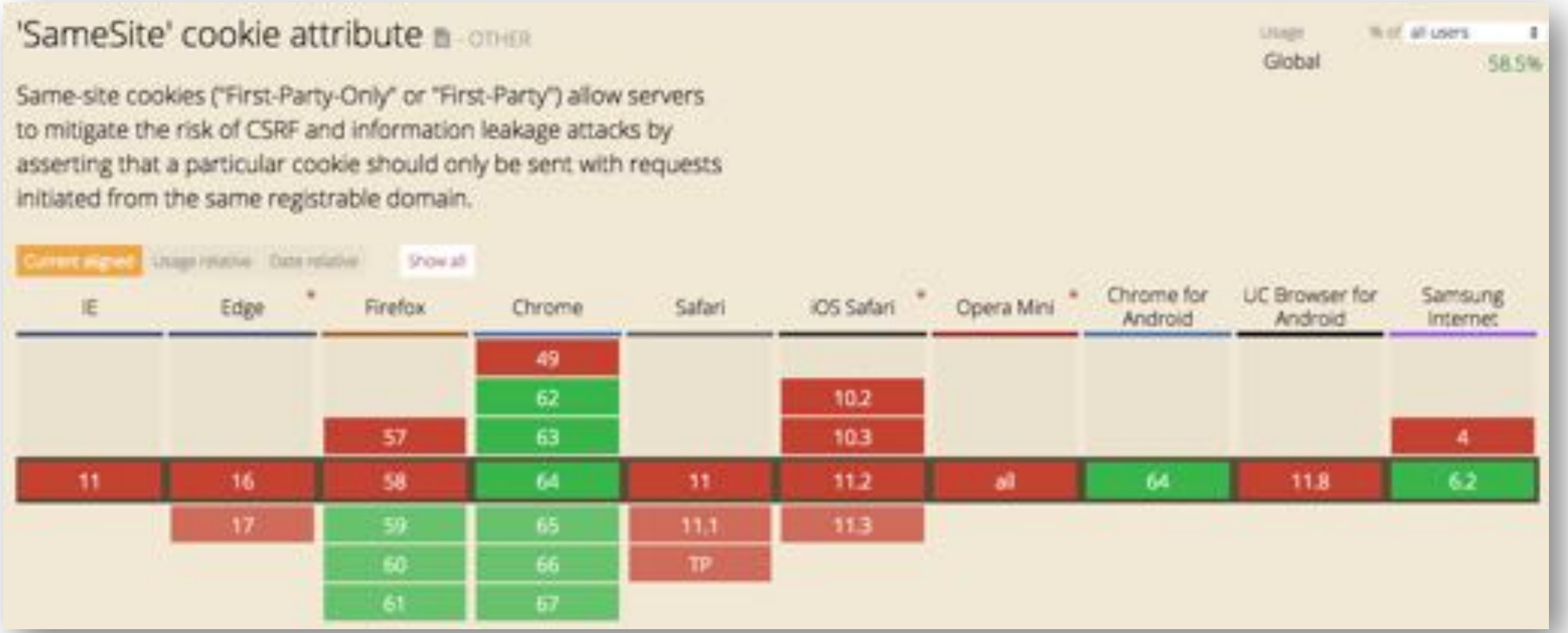
- Browsers are a quirky when sending the `Origin` header
  - It should be sent on every cross-origin request sent through `XMLHttpRequest`
  - It should be sent on every cross-origin request, except for GET and HEAD requests
- The first requirement is part of CORS (Cross-Origin Resource Sharing)
  - Since the origin header is fundamental here, it is well supported in all browsers
  - The second requirement is less crucial, and therefore support is quirky
- This makes the `Origin` header less suited as a CSRF defense
  - Except when all your calls are to an API in another origin
  - At that point, you have CORS requests, and you can easily check the `Origin` header

CSRF Defense 4: SameSite cookies




```
Set-Cookie: SSID=1234; SameSite=Strict
```

BROWSER SUPPORT FOR SAMESITE COOKIES



Overview of CSRF defenses

- Hidden form tokens
  - Requires server-side storage of CSRF tokens, which may be resource-intensive
- Transparent tokens
  - Stateless CSRF defense mechanism
  - Extremely compatible with client-side JavaScript applications (e.g. AngularJS)
- Checking the origin header
  - Useful when other context information is missing
  - Plays an important role when accessing APIs with Cross-Origin Resource Sharing (CORS)
  - Practical defense during the setup of a WebSocket connection
- SameSite cookies
  - Addresses the root of the problem, but browser support is still very limited

ANGULAR SUPPORTS TRANSPARENT TOKENS BY DEFAULT

Cross Site Request Forgery (XSRF) Protection

XSRF is an attack technique by which the attacker can trick an authenticated user into unknowingly executing actions on your website.
AngularJS provides a mechanism to coun
default, XSRF-TOKEN) and sets it as an H
cookie, your server can be assured that th
domain requests.

To take advantage of this, your server nee
GET request. On subsequent XHR reques
sure that only JavaScript running on your
verifiable by the server (to prevent the Jav
authentication cookie with a salt for adde

The name of the headers can be specifie
config-time, $http.defaults at run-time, or

In order to prevent collisions in environme
each application uses unique cookie nam

Angular's `http` has built-in support for the client-side half of this technique in its `XSRFStrategy`. The default `CookieXSRFStrategy` is turned on automatically. Before sending an HTTP request, the `CookieXSRFStrategy` looks for a cookie called `XSRF-TOKEN` and sets a header named `X-XSRF-TOKEN` with the value of that cookie.

The server must do its part by setting the initial `XSRF-TOKEN` cookie and confirming that each subsequent state-modifying request includes a matching `XSRF-TOKEN` cookie and `X-XSRF-TOKEN` header.

XSRF/CSRF tokens should be unique per user and session, have a large random value generated by a cryptographically secure random number generator, and expire in a day or two.

Your server may use a different cookie or header name for this purpose. An Angular application can customize cookie and header names by providing its own `CookieXSRFStrategy` values.

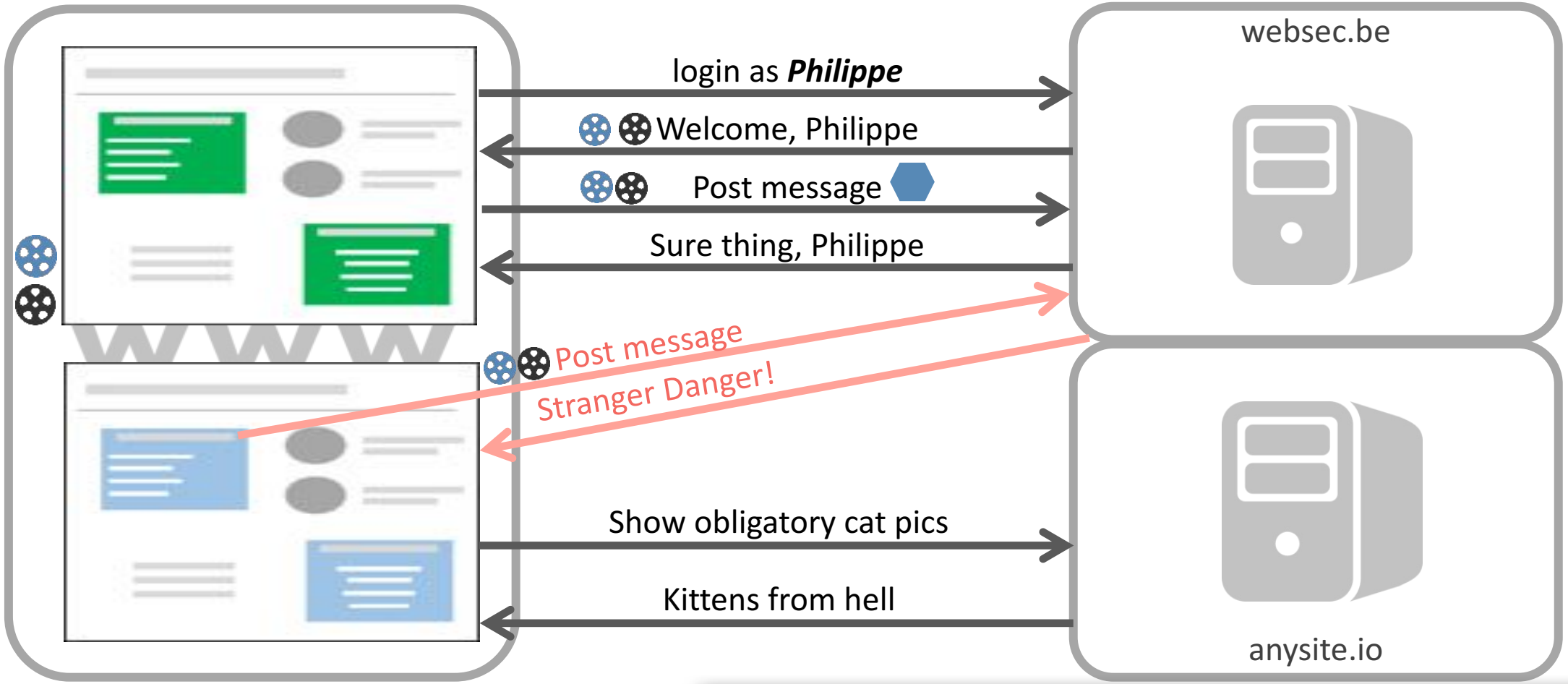
```
{ provide: XSRFStrategy, useValue: new CookieXSRFStrategy('myCookieName', 'My-Header-Name') }
```

Or you can implement and provide an entirely custom `XSRFStrategy`:

```
{ provide: XSRFStrategy, useClass: MyXSRFStrategy }
```

https://docs.angularjs.org/api/ng/service/$http
https://angular.io/docs/ts/latest/guide/security.html

CSRF Defense 2: Transparent Tokens



Cookie value is copied to a header by JavaScript code

```
POST …
Cookie: SID=123, XSRF-TOKEN=abc
X-XSRF-TOKEN: abc
```

Defending against CSRF in Angular

- Protect your application by deploying an appropriate CSRF defense
  - Angular supports transparent tokens out of the box
  - If the API is accessed over CORS, the `Origin` header is a viable alternative
- Make sure your backend is fully aware of the potential impact of CSRF
  - Enable CSRF checks for all entry points, except authentication
  - Avoid performing state-changing effects with GET requests (e.g. logout)
  - Be aware of frameworks that collate GET and POST requests
- Use the `SameSite` cookie attribute for additional security
  - Will only work if your application and backend run within the same domain

JWT

JSON Web Tokens are an open, industry standard **RFC 7519** method for representing claims securely between two parties.

http://jwt.io/


eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6IjUwNjc3YmQzLTRjZmMtNDEyZC1iNjI0LWE1MTA5NjM1ZTU0ZiJ9.eyJpc3MiOiJiZWVyc2FmZS5ldSIsInVzZXIiOjF9.LnuBPKi0wbUZBWwJVC_-FMv2QSqVLSTn-s2tL393yHo
HEADER: ALGORITHM & TOKEN TYPE
{
"alg": "HS256",
"typ": "JWT",
"kid": "50677bd3-4cfc-412d-b624-a5109635e54f"
}
PAYLOAD: DATA
{
"iss": "beersafe.eu",
"user": 1
}
VERIFY SIGNATURE
HMACSHA256(
base64UrlEncode(header) + "." +
base64UrlEncode(payload),
YourApplicationSecret
) secret base64 encoded

JWT REPRESENTS DATA, NOT THE TRANSPORT MECHANISM

- The ***cookies vs tokens*** debate can be a bit confusing
  - Cookies are a transport mechanism, just like the `Authorization` header
  - Tokens are a representation of (session) data, like a (session) identifier

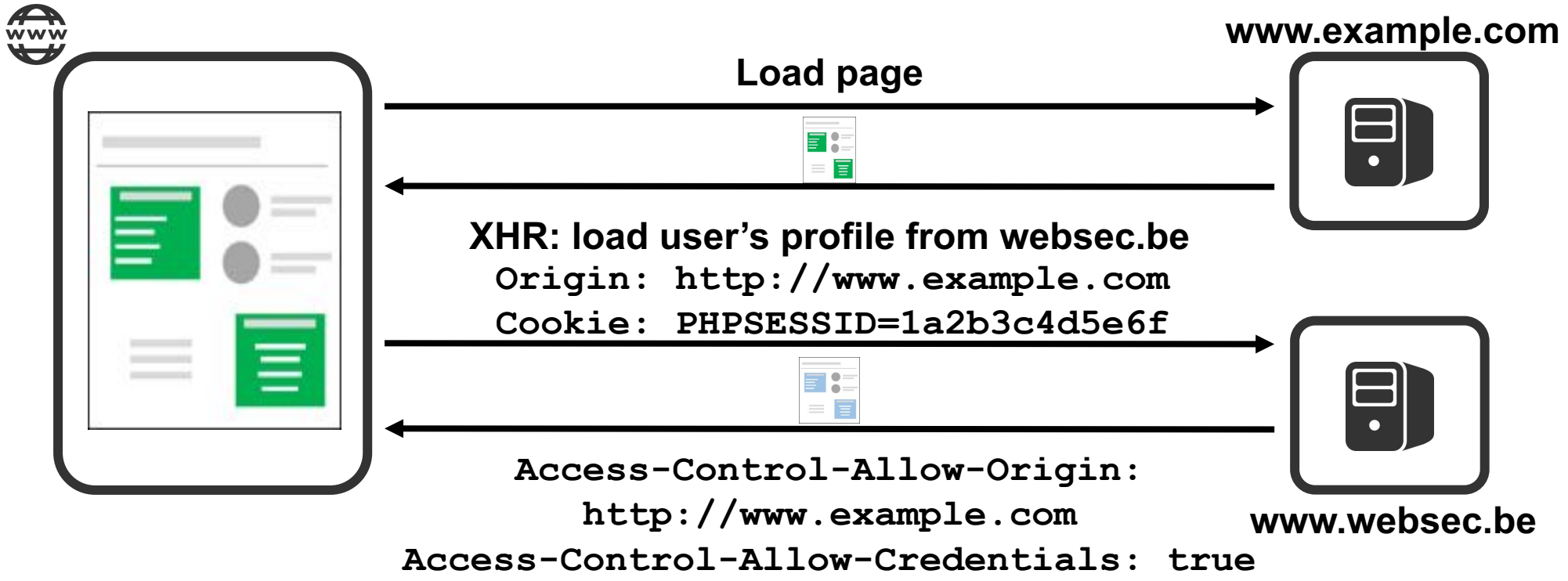
- JWT tokens can be transmitted in a cookie, or in the `Authorization` header
  - Defining how to transmit a JWT token is up to the web application
  - This choice determines the need for JavaScript support and CSRF defenses

- Modern applications typically use JWT in the `Authorization` header
  - Frontend JavaScript apps can easily put the token into the `Authorization` header
  - JWT tokens are easy to pass around between services in the backend as well

PUTTING IT ALL TOGETHER

Simple CORS Example with Credentials

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



WebSockets depend on the Origin header

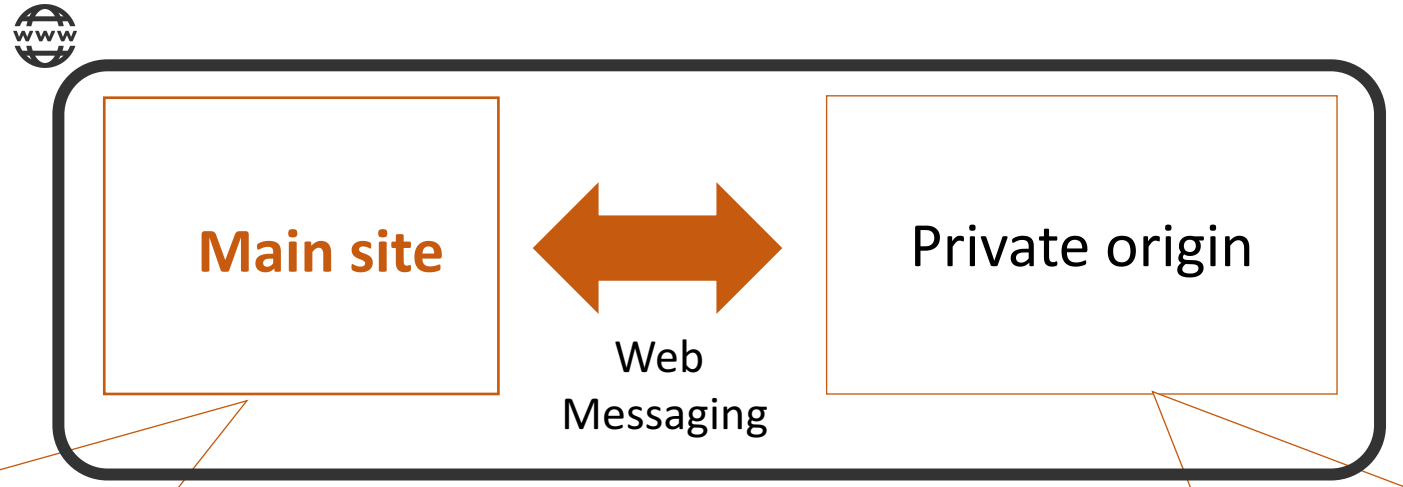
[OSSA 2015-005] Websocket Hijacking Vulnerability in Nova VNC Server (CVE-2015-0259)

Bug #1409142 reported by Josh Kleinpeter on 2015-01-09

This gives the attacker full read-write access to the VNC console of any instance recently accessed by the victim.

https://bugs.launchpad.net/nova/+bug/1409142

KEEPING SECRETS IN THE BROWSER



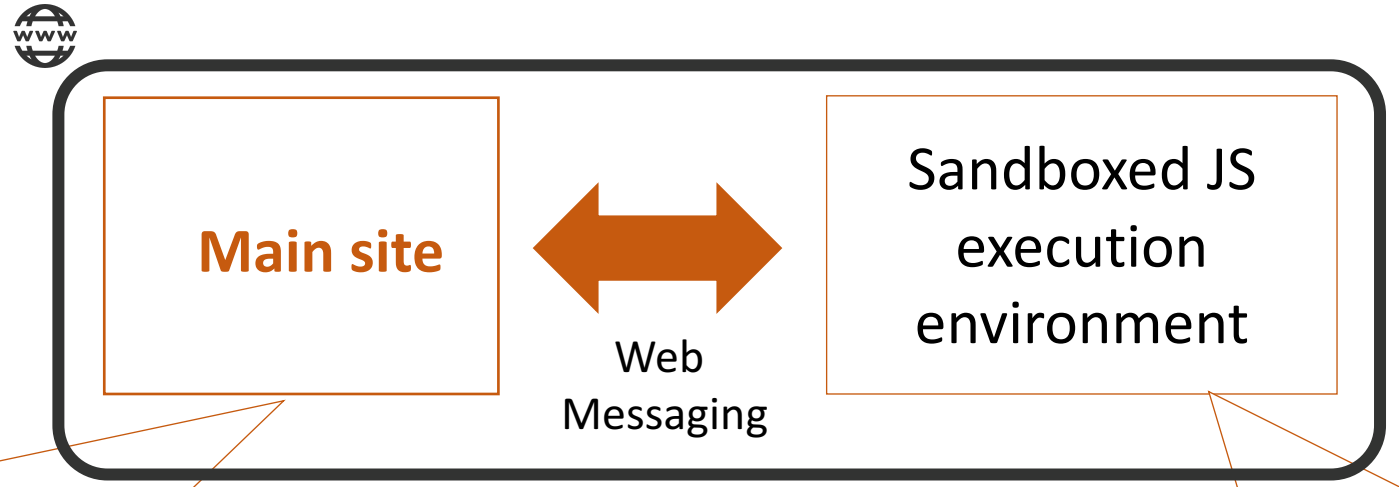
Main site

Asks the private origin to make certain requests, without having to know the API key.

Iframe with a separate origin

Only the code to keep the API key and perform API calls runs here

DOCUMENT RENDERING IN CHROMEOS



Main site

Protected with CSP to prevent injection
Delegates insecure executions to the sandboxed iframe

Sandboxed iframe

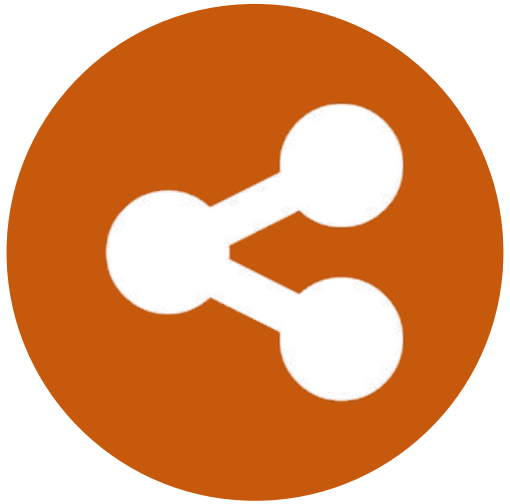
Runs in unique origin
Allowed to run JS

https://speakerdeck.com/mikewest/securing-the-client-side-devoxx-2012

Now it's up to you ...

Secure

Follow



Share