Recent Web Security Technology

Lieven Desmet – iMinds-DistriNet, KU Leuven
Lieven.Desmet@cs.kuleuven.be

SecAppDev Leuven 2016 (11/03/2016, Leuven)

DistriNet
iMinds KU LEUVEN

About myself: Lieven Desmet

@lieven_desmet

- Research manager at KU Leuven
  - (Web) Application Security
- Active participation in OWASP
  - Board member of the OWASP Belgium Chapter
  - Co-organizer of the OWASP AppSec EU Conferences
- Program director at SecAppDev

DistriNet iMinds KU LEUVEN

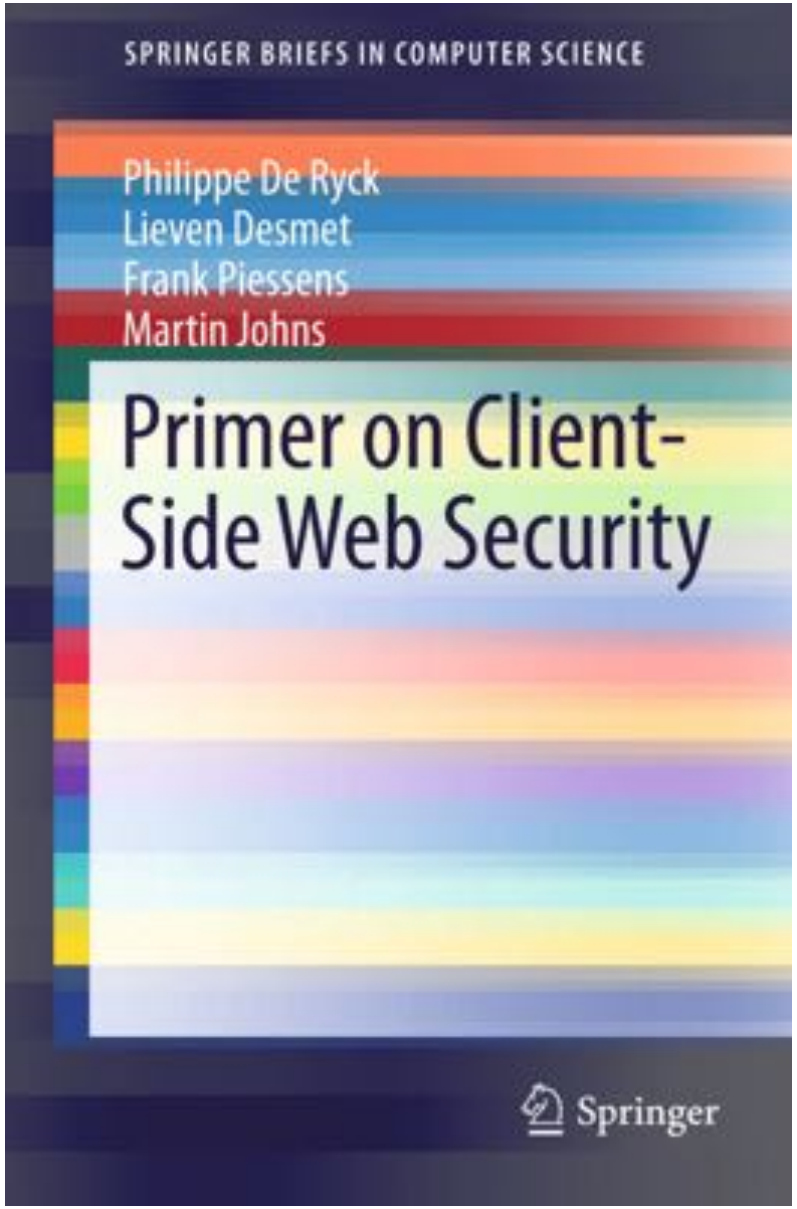
iMinds-DistriNet, KU Leuven

- Headcount:
  - 10 professors
  - 65 researchers
- Research Domains
  - Secure Software
  - Distributed Software
- Academic and industrial collaboration in 30+ national and European projects

https://distrinet.cs.kuleuven.be

Primer on Client-Side Web Security

- Covers the landscape of client-side Web security
  - State-of-the-art in web security
  - State-of-practice on the Web
  - Recent research and standardization activities
  - Security best practices per category



Recent Web Security Technology

Server-side security policies, enforced by the browser

DistriNet
iMinds KU LEUVEN

Sans Top 25 - OWASP Top 10

Rank	Score	ID	Name
[1]	93.8	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[2]	83.3	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[3]	79.0	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[4]	77.7	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[5]	76.9	CWE-306	Missing Authentication for Critical Function
[6]	76.8	CWE-862	Missing Authorization
[7]	75.0	CWE-798	Use of Hard-coded Credentials
[8]	75.0	CWE-311	Missing Encryption of Sensitive Data
[9]	74.0	CWE-434	Unrestricted Upload of File with Dangerous Type
[10]	73.8	CWE-807	Reliance on Untr
[11]	73.1	CWE-250	Execution with U
[12]	70.1	CWE-352	Cross-Site Requ
[13]	69.3	CWE-22	Improper Limitat
[14]	68.5	CWE-494	Download of Cod
[15]	67.8	CWE-863	Incorrect Author
[16]	66.0	CWE-829	Inclusion of Func
[17]	65.5	CWE-732	Incorrect Permis
[18]	64.6	CWE-676	Use of Potentiall
[19]	64.1	CWE-327	Use of a Broken
[20]	62.4	CWE-131	Incorrect Calcul
[21]	61.5	CWE-307	Improper Restric
[22]	61.1	CWE-601	URL Redirection
[23]	61.0	CWE-134	Uncontrolled For
[24]	60.3	CWE-190	Integer Overflow
[25]	59.9	CWE-759	Use of a One-Way Hash without a Salt

Focus on vulnerabilities and logical flaws in the code, and server-side mitigations

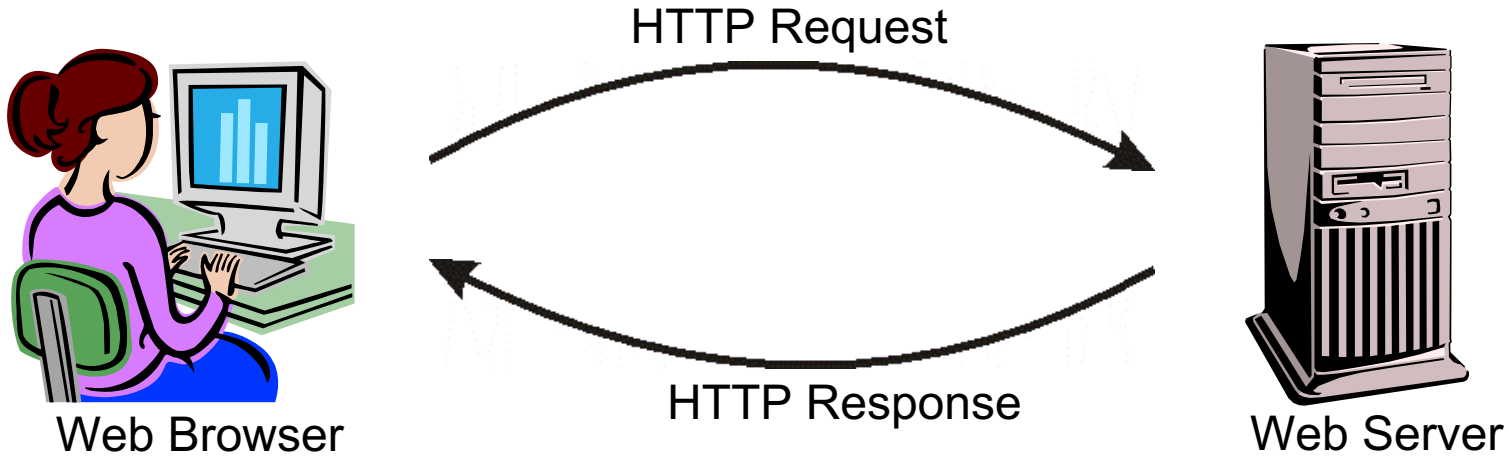
This talk focuses on infrastructural support as a complementary line of defense

- 2013 (New)
ession Management
nces
A5 – Security Misconfiguration
A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control
A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Known Vulnerable Components
A10 – Unvalidated Redirects and Forwards

SANS

DistriNet iMinds KU LEUVEN

Recent security technology on the web



Overview

- Introduction
- #1 Securing browser-server communication
- #2 Mitigating script injection attacks
- #3 Framing content securely
- Example security architecture: Combining CSP & Sandbox
- Wrap-up

Introduction

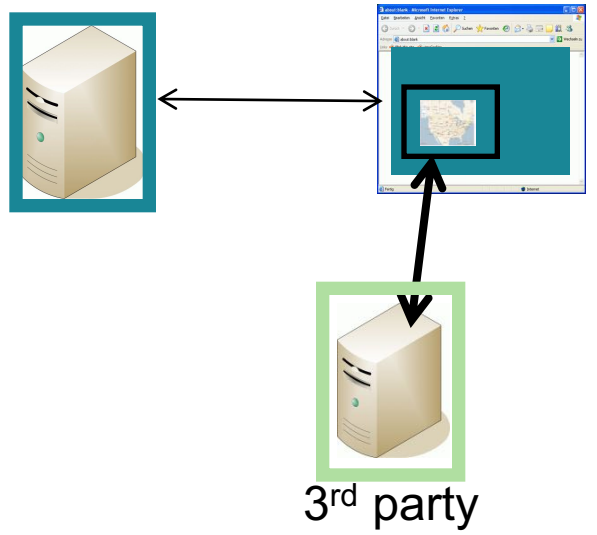
Recap: Web's Security Model

- Basic security policy for the web:
  - Same-Origin Policy
- What does it mean for scripts running on your page?
- What does it mean for frames included in your page?

Two basic composition techniques

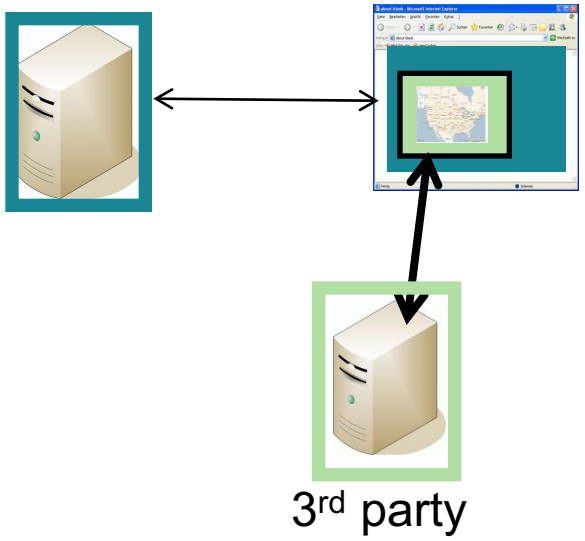
Script inclusion

```
<html><body>
…
<script src="http://3rdparty.com/script.js"></script>
…
</body></html>
```



Iframe integration

```
<html><body>
…
<iframe src="http://3rdparty.com/frame.html"></iframe>
…
</body></html>
```



State of practice metrics

- Assessment of the most popular European Union websites
  - Top 1,000,000 websites from Alexa raking
  - Filter top 1,000 websites of 28 member states
  - Result: 23,050 European websites

STREWS

DistriNet

iMinds KU LEUVEN

ccTLD	Country	ccTLD	Country	ccTLD	Country	ccTLD	Country
at	Austria	ee	Estonia	ie	Ireland	pl	Poland
be	Belgium	es	Spain	it	Italy	pt	Portugal
bg	Bulgaria	fi	Finland	lt	Lithuania	ro	Romania
cy	Cyprus	fr	France	lu	Luxembourg	se	Sweden
cz	Czech Republic	gr	Greece	lv	Latvia	si	Slovenia
de	Germany	hr	Croatia	mt	Malta	sk	Slovakia
dk	Denmark	hu	Hungary	nl	Netherlands	uk	United Kingdom

Longitudinal study

- Crawl up to 200 pages per website
  - Use a headless browser (PhantomJS)
  - Capture all data and headers sent by server
- Compare two datasets:
  - September 2013
  - September 2015

Time	# of websites	# of webpages	avg. # of pages/site
September 2013	20,147	3,499,080	174
September 2015	18,074	2,992,395	166

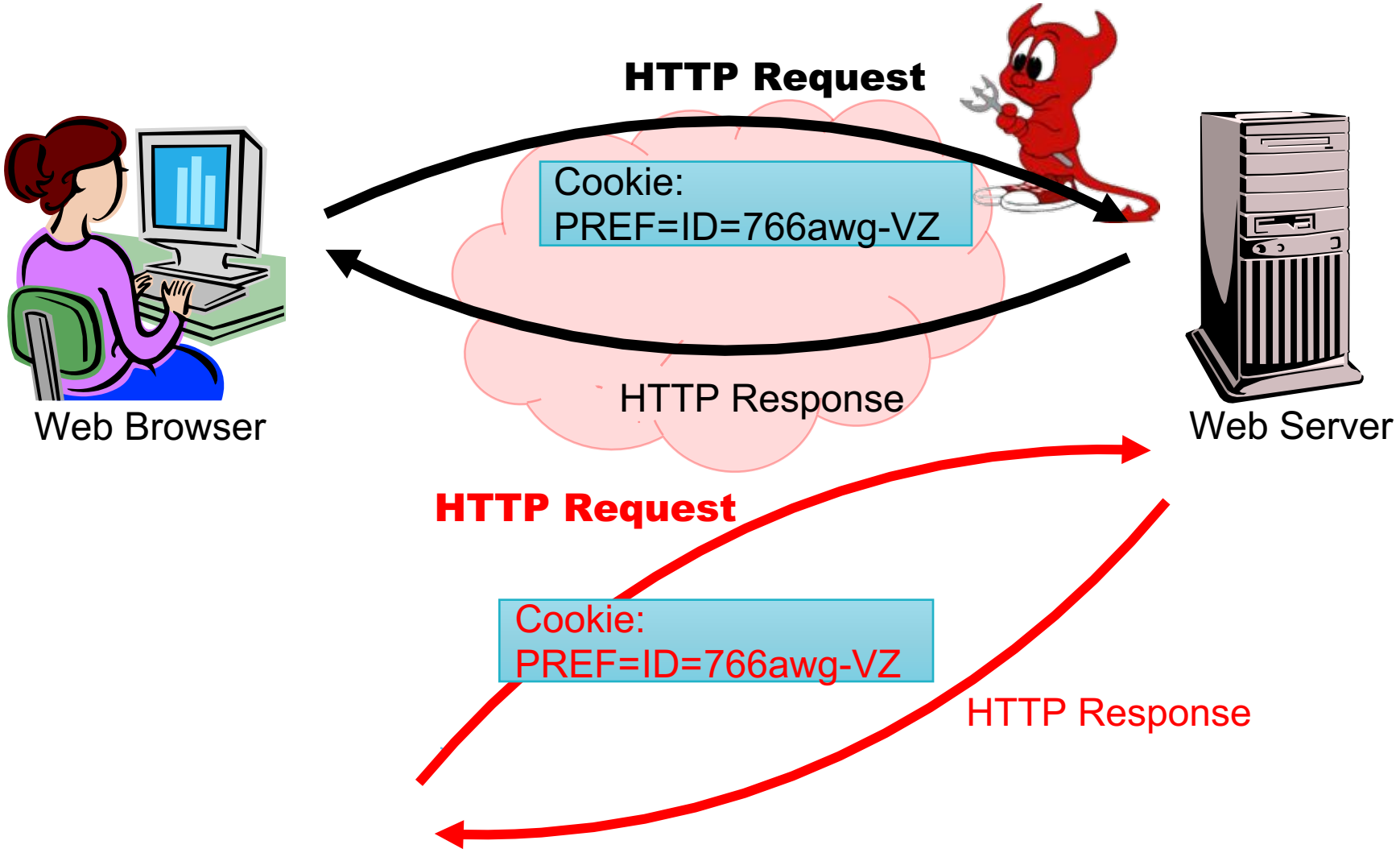
Notice: we removed the websites with less than 50 successfully crawled pages from our dataset

#1 Securing browser-server communication

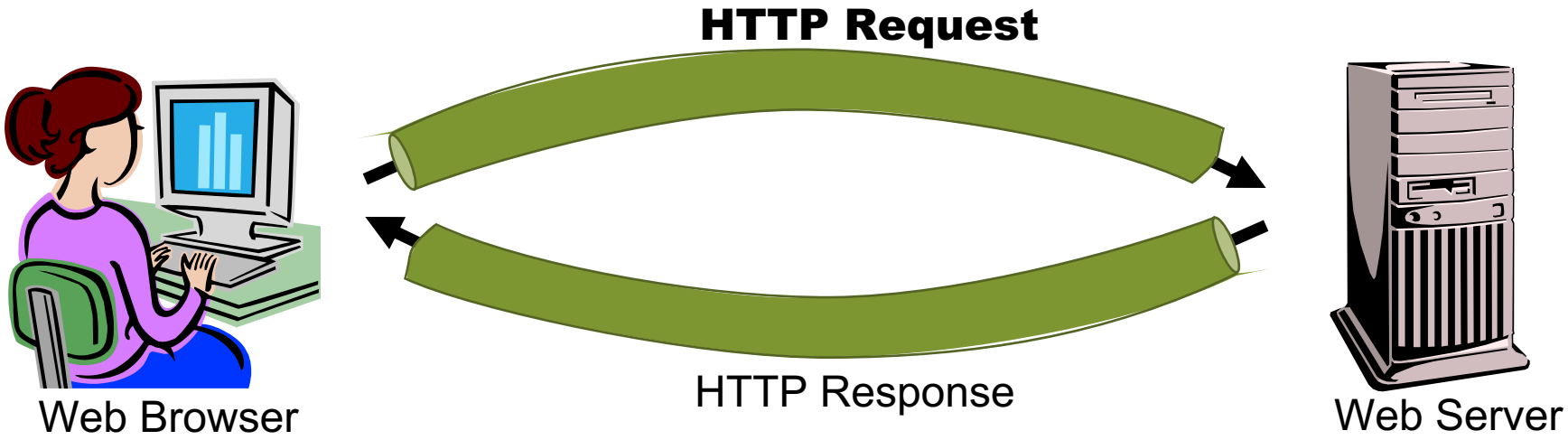
Overview

- Attacks:
  - Session hijacking
  - SSL Stripping
- Countermeasures:
  - Use of SSL/TLS
  - Secure flag for session cookies
  - HSTS header
  - Public Key Pinning

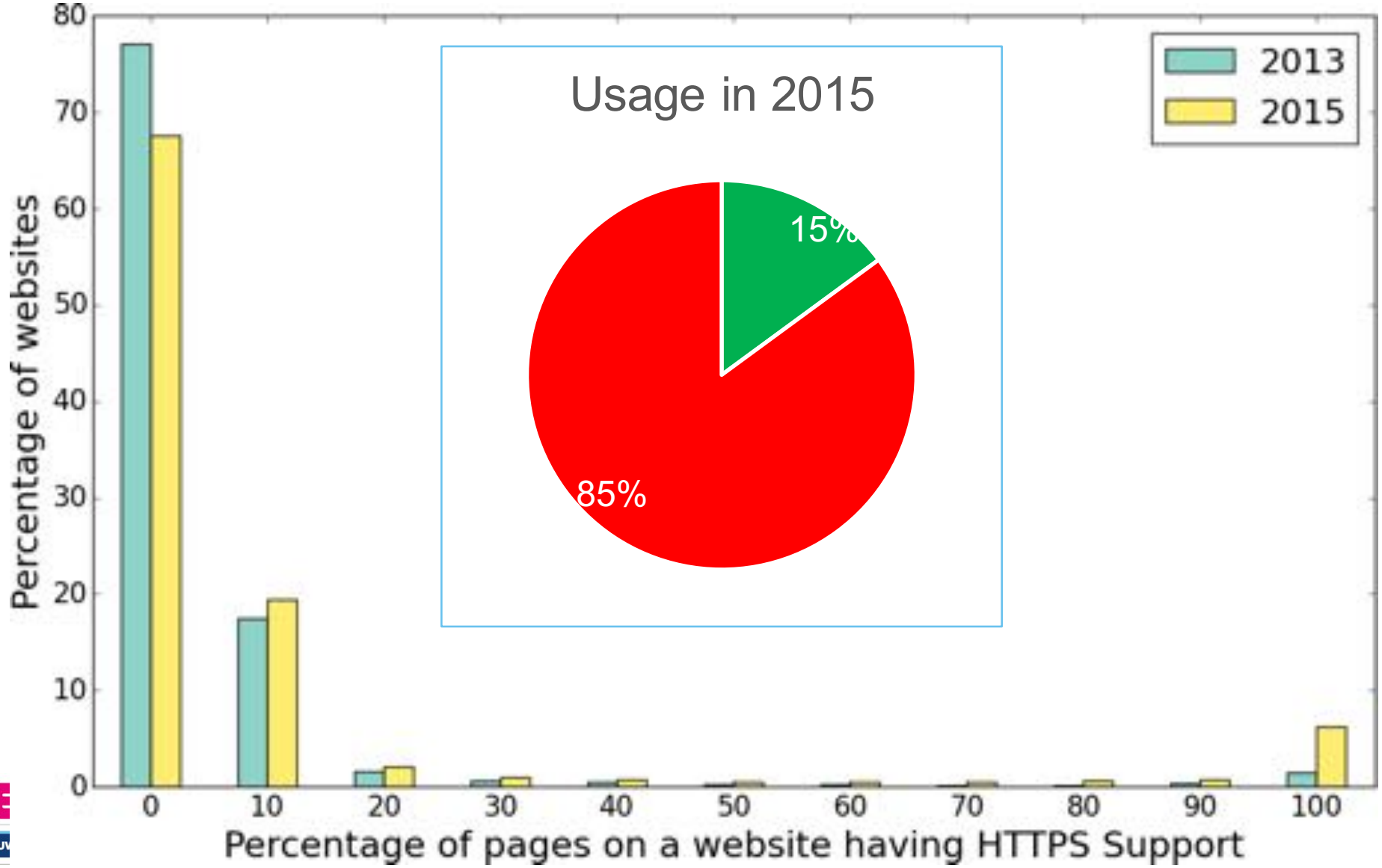
Network attacks: Session hijacking



HTTPS to the rescue...



HTTPS: State of practice



Availability of TLS (and SNI)

Server Name Indication

OTHER

An extension to the TLS computer networking protocol by which a client indicates which hostname it is attempting to connect to at the start of the handshaking process.

Global	96.27% + 0.88% = 97.15%
Belgium	97.81% + 0.5% = 98.3%

Current aligned | Usage relative | Show all

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			45						
8			46					4.3	
9			47					4.4	
10		43	48			8.4		4.4.4	
11	13	44	49	9	35	9.2	8	47	47
	14	45	50	9.1	36	9.3			
		46	51		37				
		47	52						

Notes | Known issues (0) | Resources (2) | Feedback

1 Only supported on Windows Vista or above (not Windows XP)

Problem cured?

- TLS usage statistics (for popular websites!):
  - 67.5% of the websites don't use TLS at all
  - Only 6.5% of the websites are using TLS for at least half of their pages
- Remaining problems:
  - Mixed use of HTTPS/HTTP and session cookies
  - Mixed content websites
  - SSL Stripping attacks

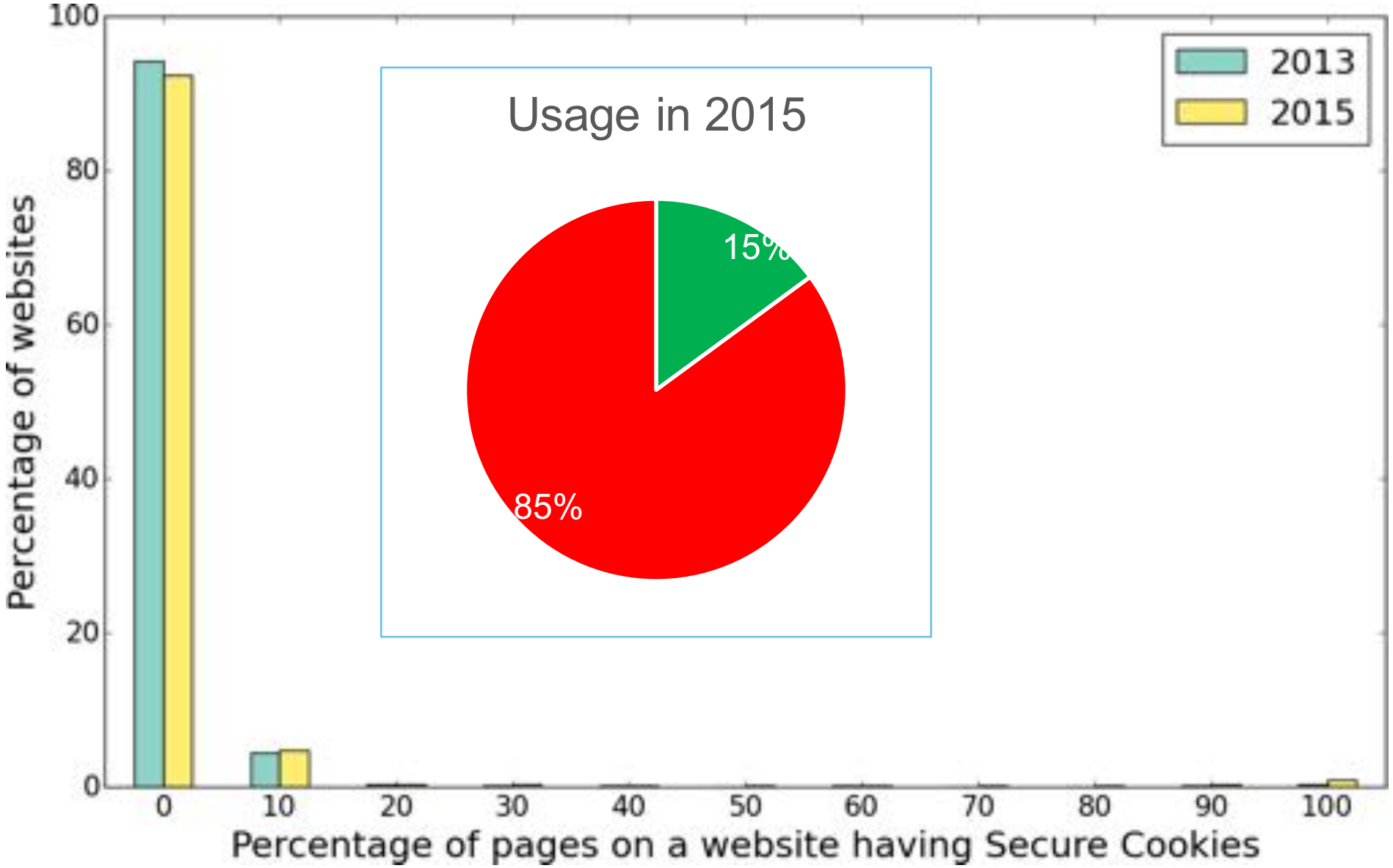
Mixed use of HTTPS/HTTP

- Cookies are bound to domains, not origins
- By default, cookies are sent both over HTTPS and HTTP
- Any request to your domain over HTTP leaks the (session) cookies…

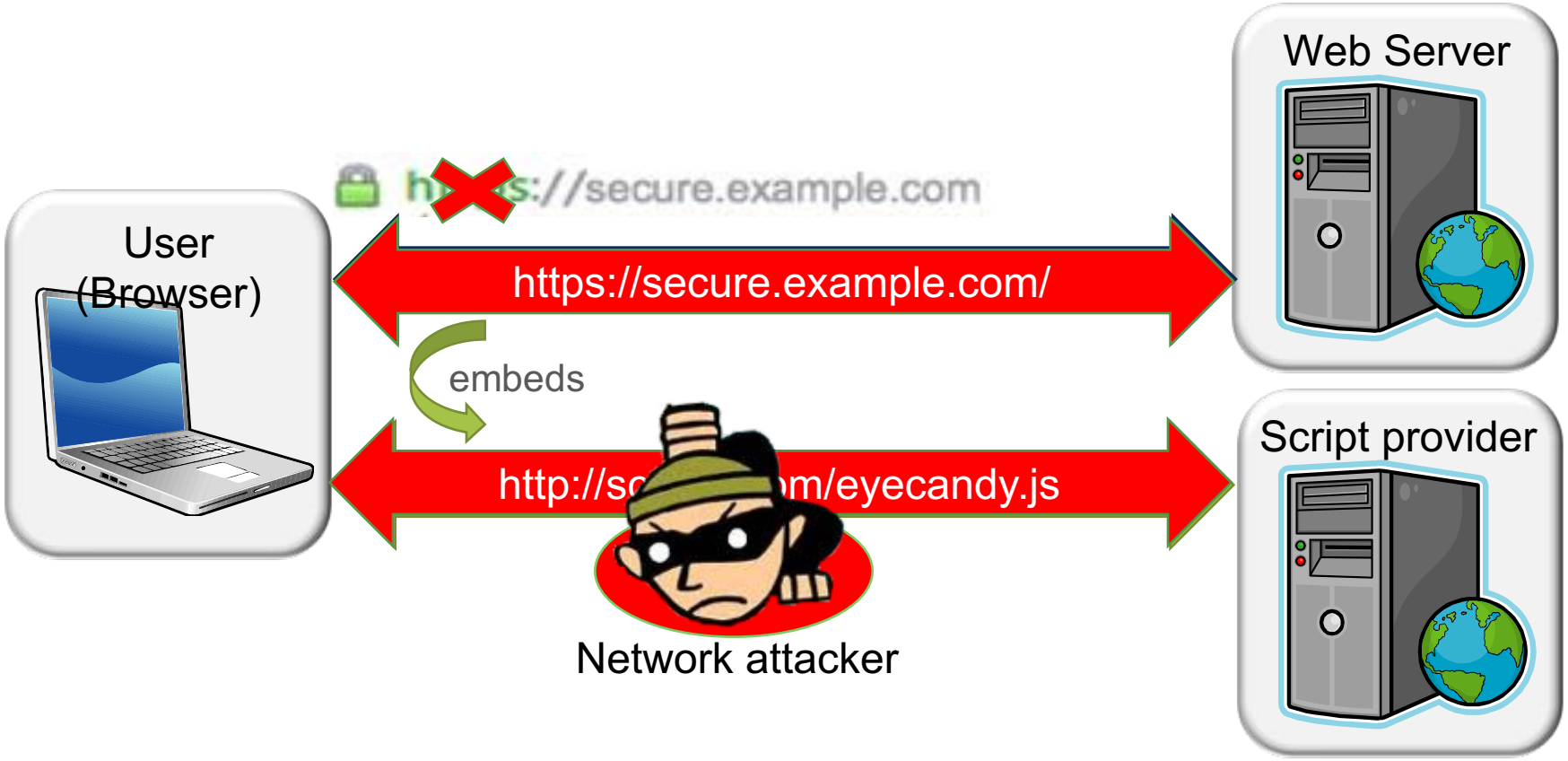
Secure flag for cookies

- Issued at cookie creation (HTTP response)
  - Set-Cookie: PREF=766awg-VZ; Domain=yourdomain.com; Secure
- If set, the cookie is only sent over an encrypted channel
- Should be enabled by default for your session cookies!

Secure cookies: State of practice



Mixed content inclusions:
TLS-enabled sites under attack



Source: *Ping Chen et. al.* A Dangerous Mix: Large-scale analysis of mixed-content websites. *ISC 2013*

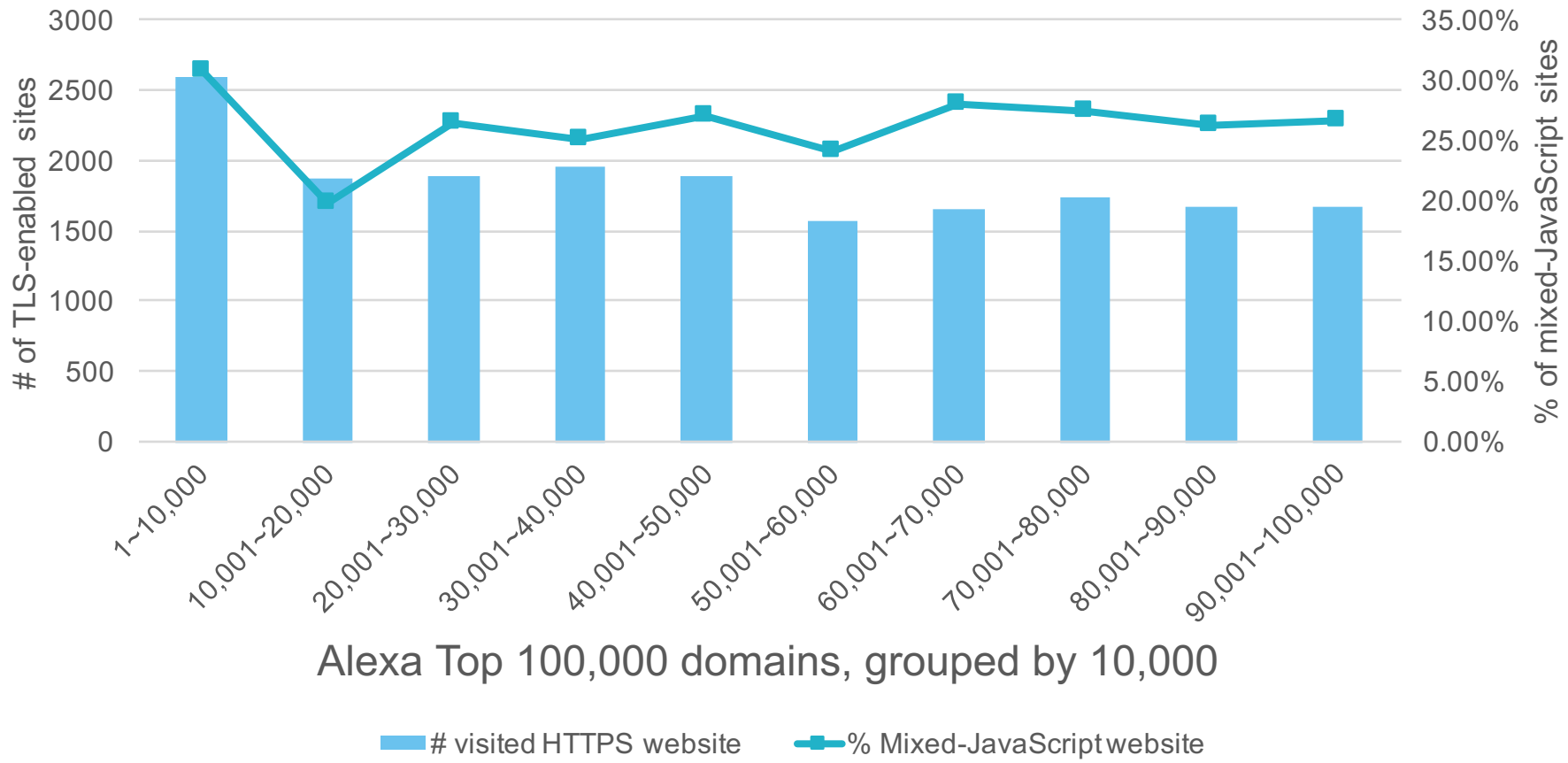
Mixed content inclusions:
Large scale assessment of the state-of-practice

- Alexa Top 100,000 domains
- Crawled over 480,000 pages belonging to the Alexa top 100,000
- Discovered:
  - 18,526 TLS-protected sites
  - 7,980 sites have mixed content (43% of the sites)
  - 150,179 scripts are included over HTTP (26% of the sites)

DistriNet iMinds KU LEUVEN

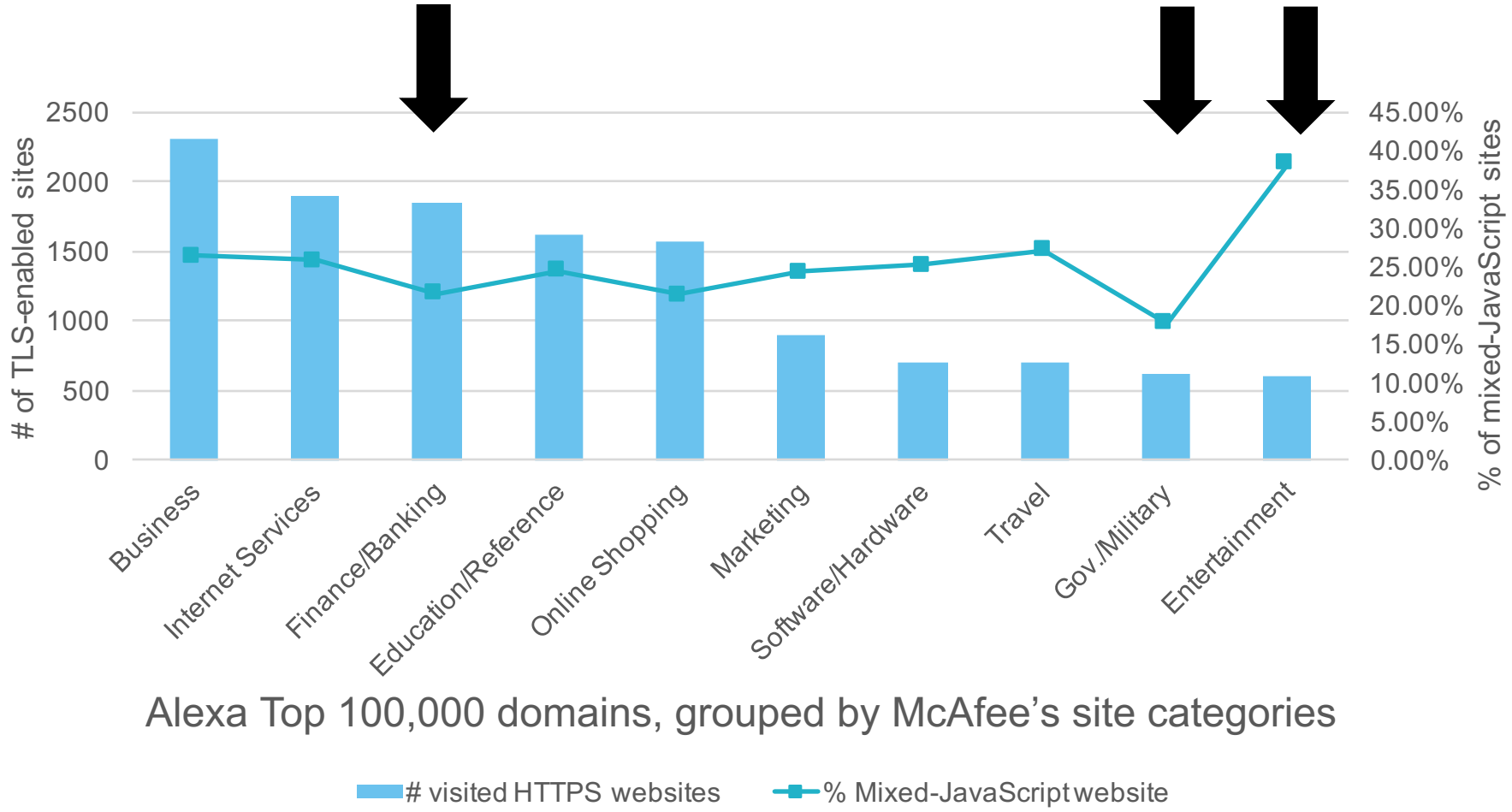
Source: *Ping Chen et. al.* A Dangerous Mix: Large-scale analysis of mixed-content websites. *ISC 2013*

Distribution of mixed-JavaScript sites
across the top Alexa Top 100,000



Source: *Ping Chen et. al.* A Dangerous Mix: Large-scale analysis of mixed-content websites. *ISC 2013*

Distribution of mixed-JavaScript sites
across Top 10 site categories (McAfee's web database)



Source: *Ping Chen et. al.* A Dangerous Mix: Large-scale analysis of mixed-content websites. *ISC 2013*

Browsers allowing mixed-content

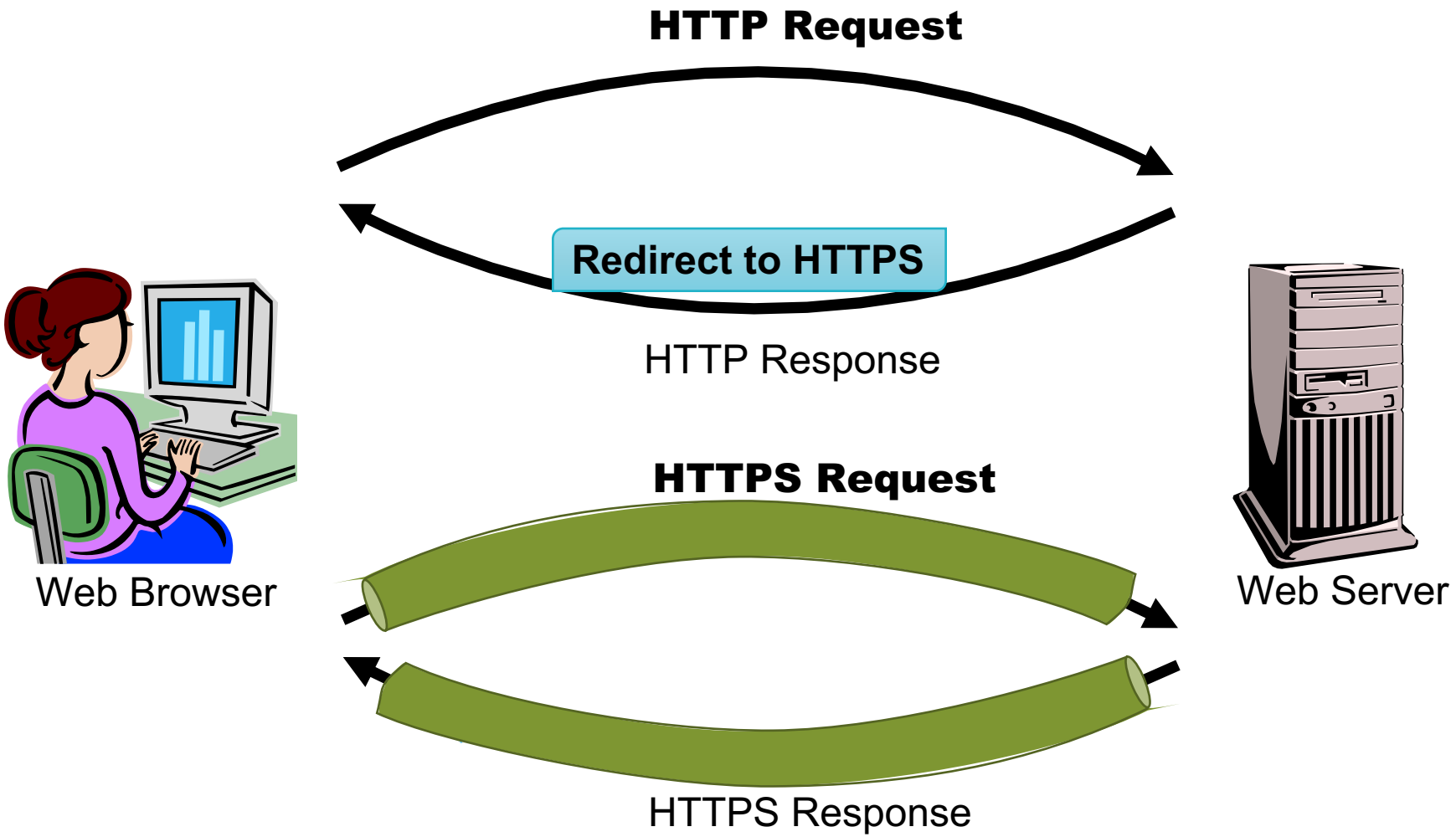
- Browsers are more and more blocking mixed-content
  - Mobile browsers were lagging behind

Browser	Images	CSS	Scripts	XHR	WebSockets	Frames
Android browser 4.4.x	Yes	Yes	Yes	Yes	Yes	Yes
Chrome 33	Yes	No	No	Yes	Yes	No
Firefox 28	Yes	No	No	No	No	No
Internet Explorer 11	Yes	No	No	No	No	No
Safari 7	Yes	Yes	Yes	Yes	Yes	Yes

Source: Qualys, February 2014

  - Safari 9 and Android 5.x block mixed content

HTTP to HTTPS bootstrapping

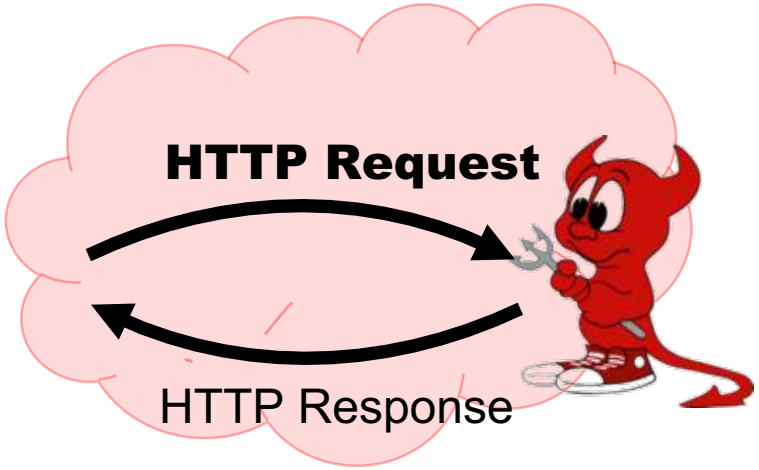


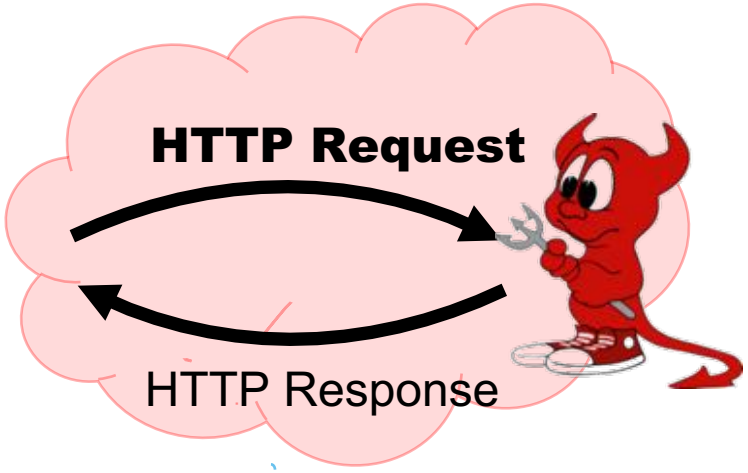
HTTP to HTTPS bootstrapping

- HTTP 301/302 response
  - Location header redirects browser to the resource over HTTPS
  - Location: https://mysite.com/
- Meta refresh
  - Meta-tag in HEAD of HTML page
  - <meta http-equiv="refresh" content="0;URL='https://mysite.com/'">
- Via JavaScript
  - document.location = “https://mysite.com”

Network attacks: SSL Stripping

Web Browser





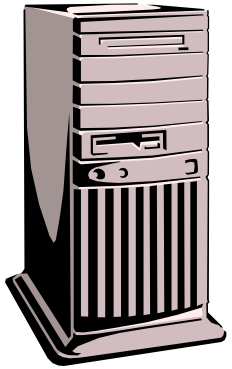
HTTP Request

HTTP Response

Redirect to HTTPS

HTTPS Request

HTTPS Response



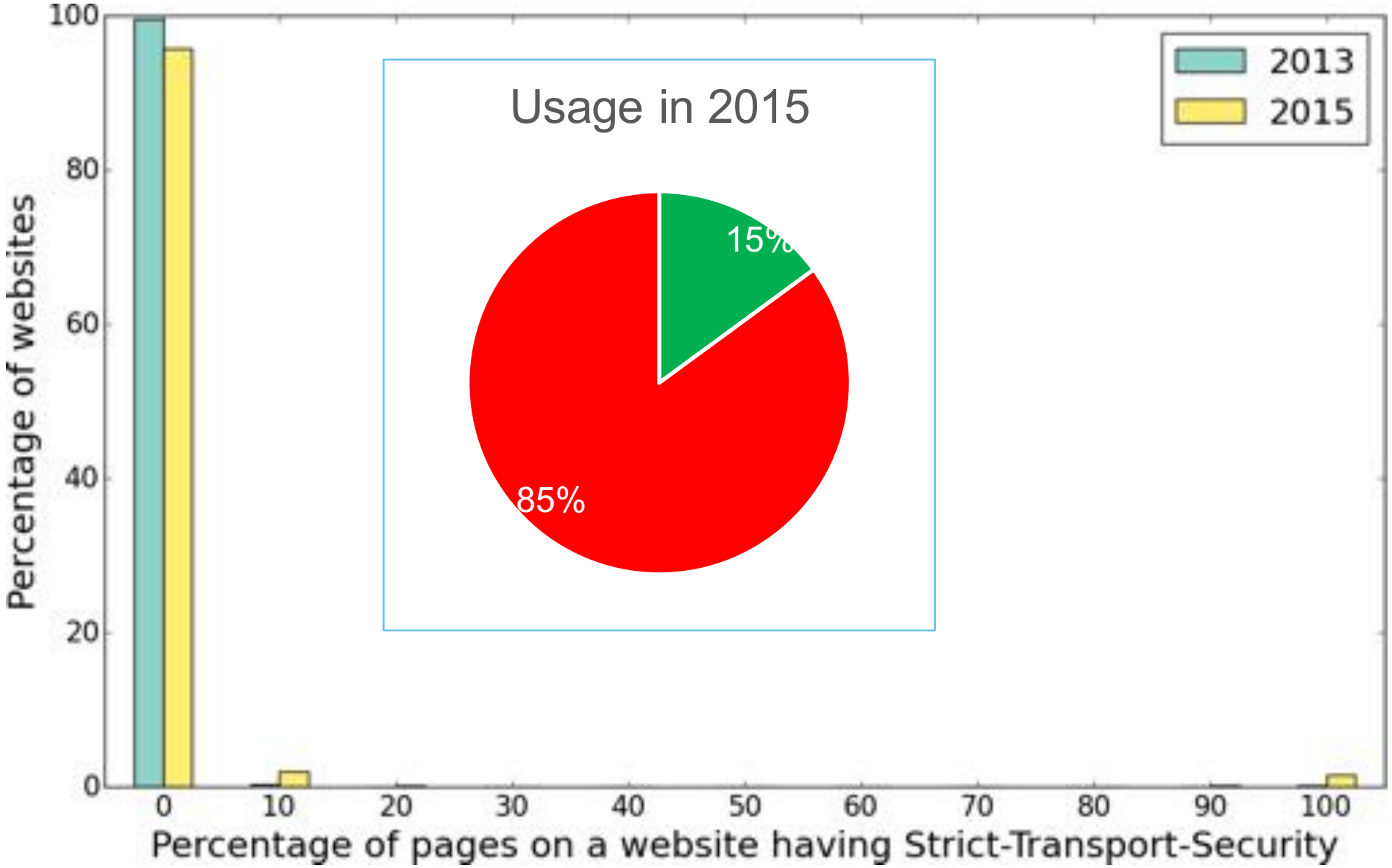
Web Server

Moxie Marlinspike, BlackHat DC 2009

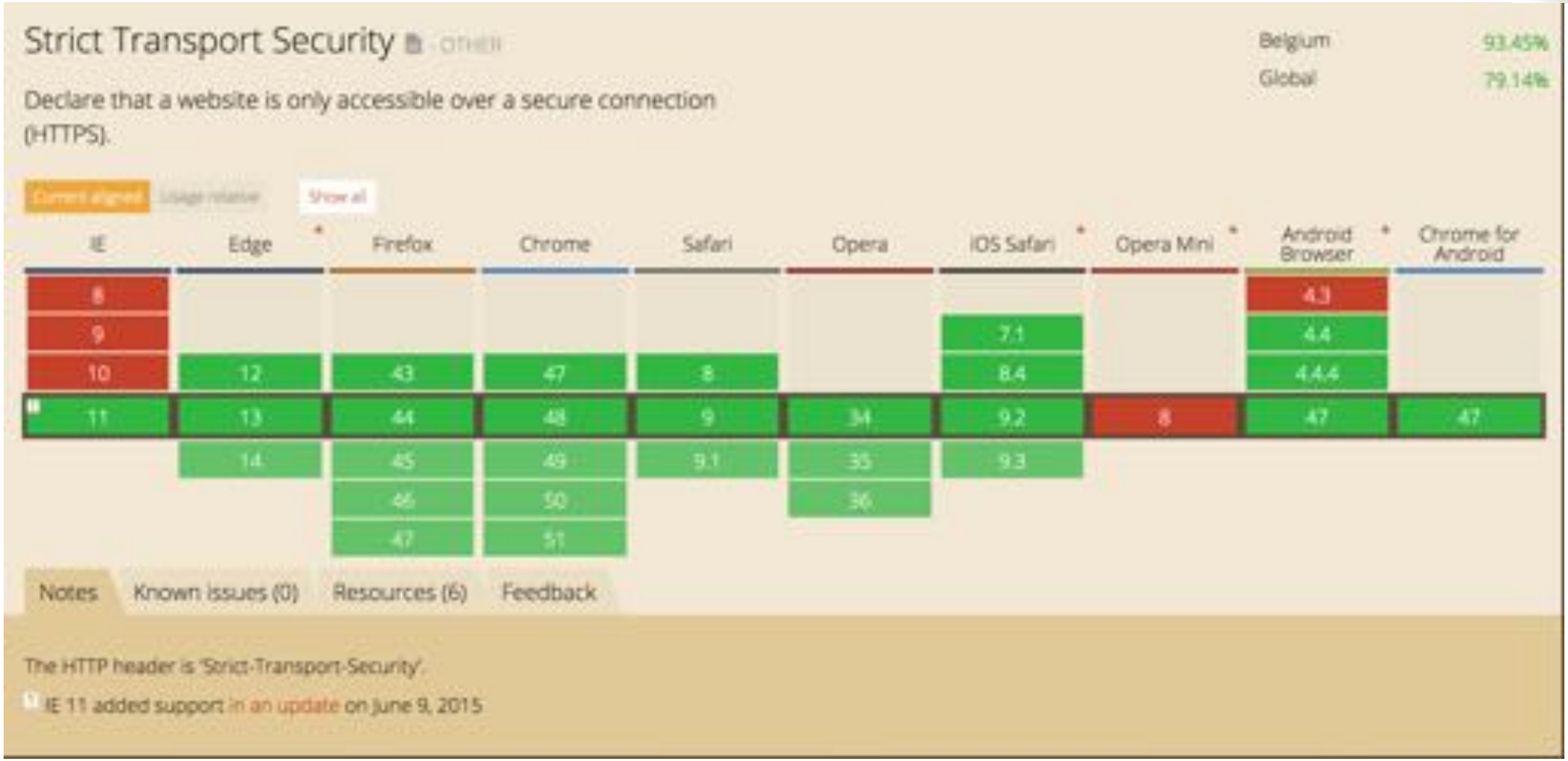
Strict Transport Security (HSTS)

- Issued by the HTTP response header
  - Strict-Transport-Security: max-age=60000
- If set, the browser is instructed to visit this domain only via HTTPS
  - No HTTP traffic to this domain will leave the browser
- Optionally, also protect all subdomains
  - Strict-Transport-Security: max-age=60000; includeSubDomains
- HSTS Browser Preloading:
  - https://hstspreload.appspot.com/

HSTS: State of practice



HSTS: availability in browsers



But can I trust the CAs ?

- Comodo (March 2011)
  - 9 fraudelent SSL certificates
- Diginotar (July 2011)
  - Wildcard certificates for Google, Yahoo!, Mozilla, WordPress, …
- Breaches at StartSSL (June 2011) and GlobalSign (Sept 2012) reported unsuccessful
- …

Public Key Pinning (HPKP)

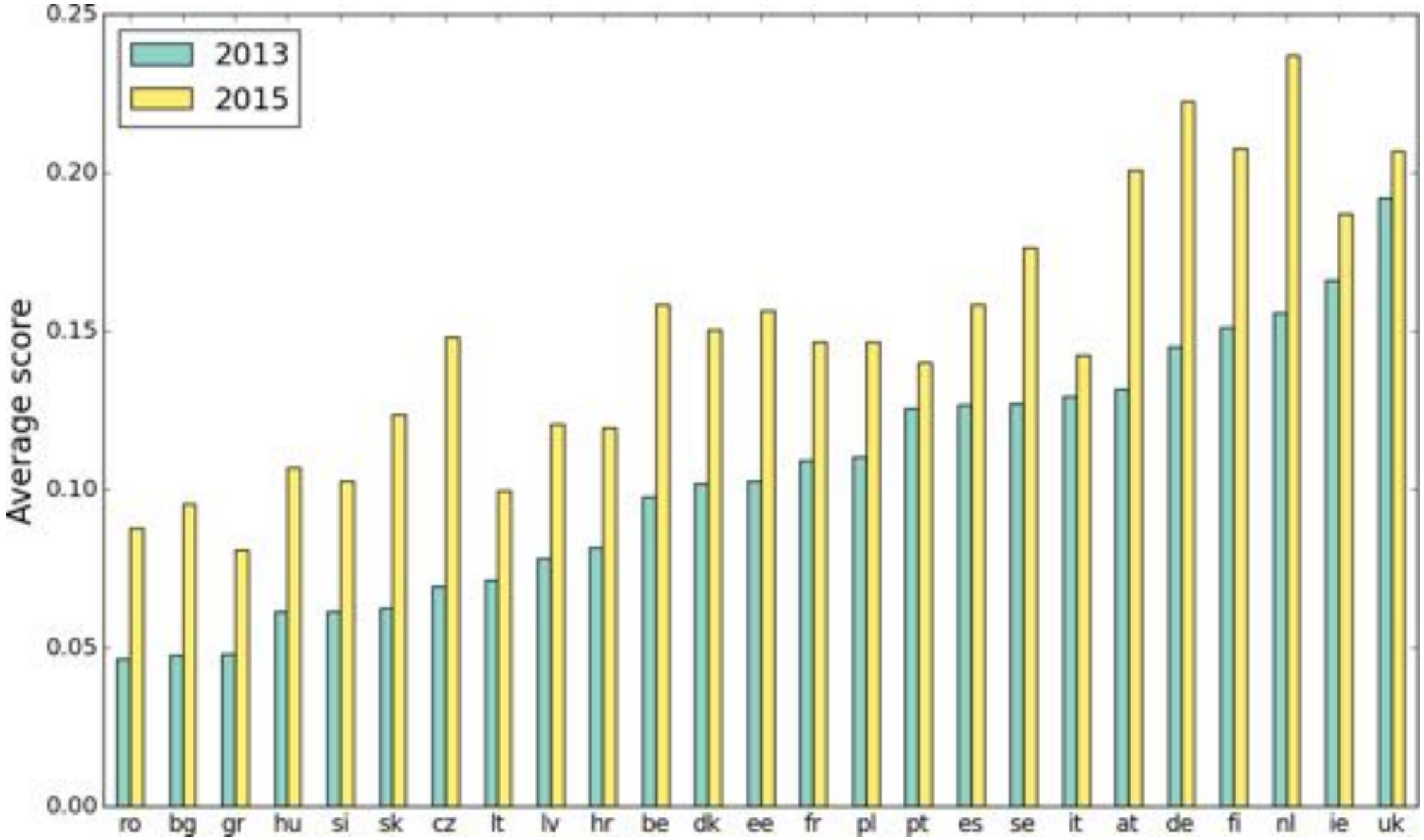
- Issued as HTTP response header
  - Public-Key-Pins: max-age=2592000;
    pin-sha256="E9CZ9INDbd+2eRQozYqqbQ2yXLVKB9+xcprMF+44U1g=";
    pin-sha256="LPJNul+wow4m6DsqxbninhsWHlwfp0JecwQzYpOLmCQ=";
    report-uri="http://example.com/pkp-report"
  - Freezes the certificate by pushing a fingerprint of (parts of) the certificate chain to the browser
  - Options: max-age, includeSubdomains, report-uri

HPKP: state-of-practice

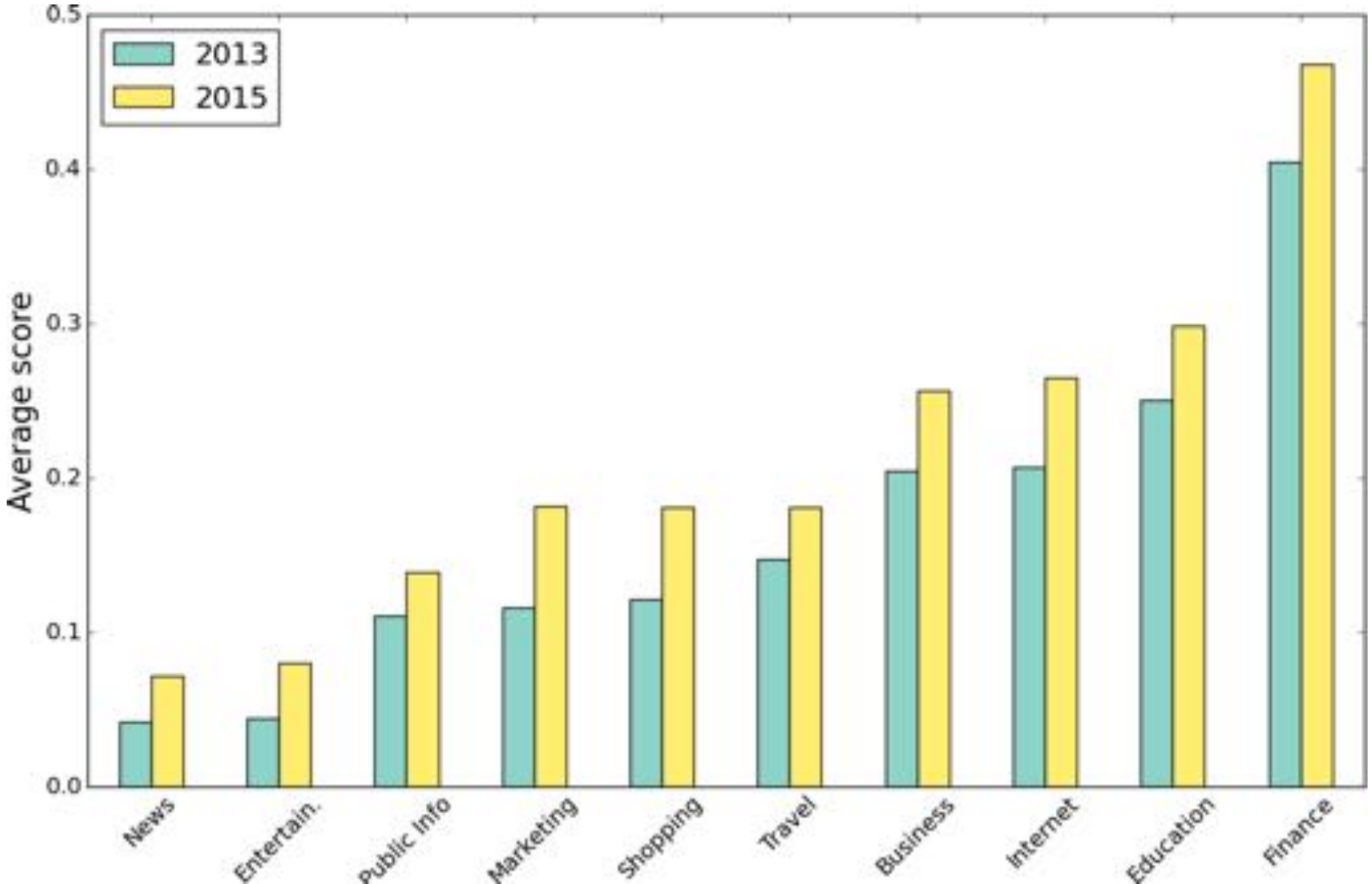
Recap: Securing browser-server communication

- Use of TLS
  - be aware of mixed-content inclusions!
- Secure flag for cookies
  - to protect cookies against leaking over HTTP
- HSTS header
  - to force TLS for all future connections
- Public Key Pinning
  - to protect against fraudulent certificates

Secure Communication Score (1)



Secure Communication Score (2)



#2 Mitigating script injection attacks

Overview

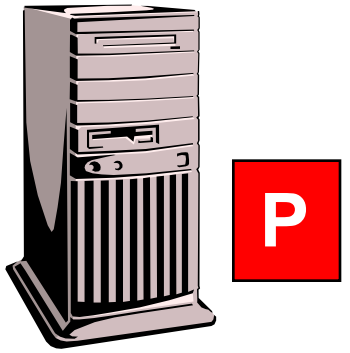
- Attack:
  - Cross-Site Scripting (XSS)
- Countermeasures:
  - HttpOnly flag for session cookies
  - X-Content-Type-Options header
  - Content Security Policy (CSP)
  - Subresource Integrity header

Example: Stored or persistent XSS

Attacker

HTTP request injecting a script
into the persistent storage of the vulnerable server

HTTP response



Vulnerable server

Regular http request

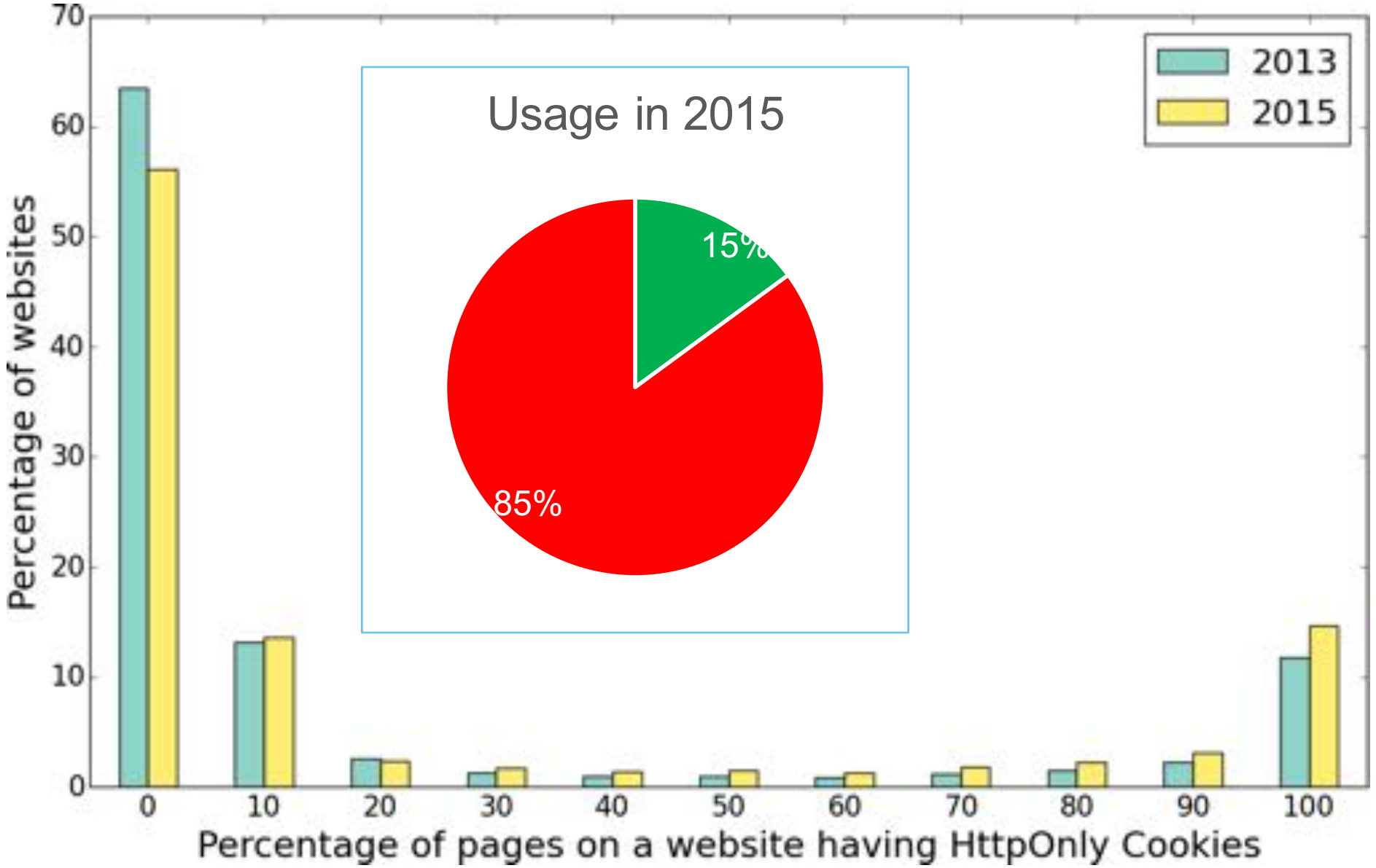
Victim

Http response containing
script as part of executable content

HttpOnly flag for cookies

- Issued at cookie creation (HTTP response)
  - Set-Cookie: PREF=766awg-VZ; Domain=yourdomain.com; Secure; HttpOnly
- If set, the cookie is not accessible via DOM
  - JavaScript can not read or write this cookie
- Mitigates XSS impact on session cookies
  - Protects against hijacking and fixation
- Should be enabled by default for your session cookies!

HttpOnly: state-of-practice



Misinterpretation of content

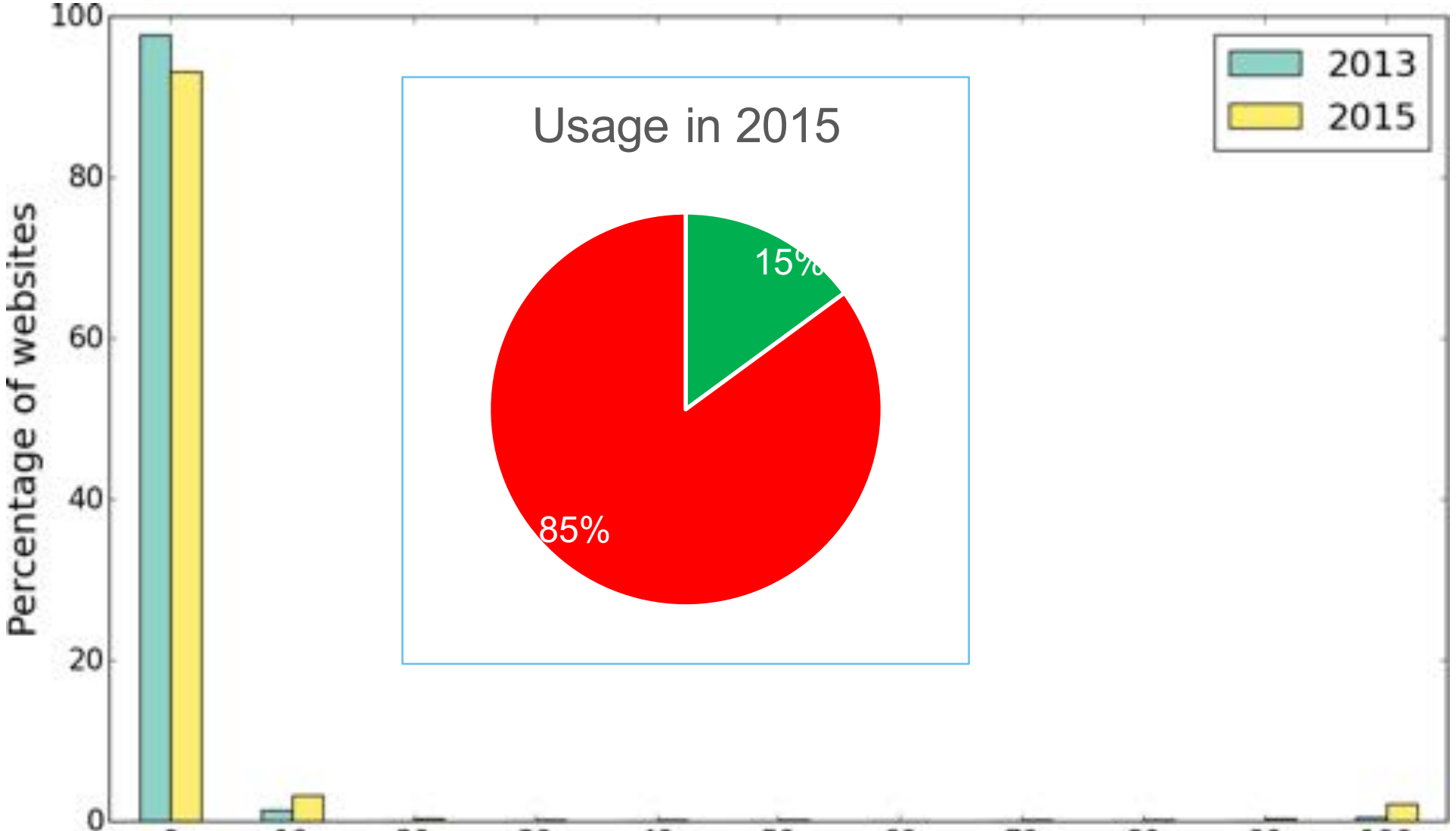
- Browsers are very relax in how content get processed
- To detect how the content may be displayed/executed, browser try to detect the content type
- Attackers can confuse the browser (eg. by sending scripts as images)
  - For the server, the resources are harmless images
  - For the client, the resources are interpreted as scripts

X-Content-Type-Options

- To disable this ‘automatic sniffing’ behavior, browser can use:
  - X-Content-Type-Options: nosniff

- Best practice for all resources:
  - Explicit MIME content types on server
  - Use X-CTO to disable client-side sniffing

X-Content-Type-Options: State of practice



Content Security Policy (CSP)

- Issued as HTTP response header
  - Content-Security-Policy: script-src 'self'; object-src 'none'
- Specifies which resources are allowed to be loaded as part of your page
- Extremely promising as an additional layer of defense against script injection

CSP set of directives

- There are a whole set of directives
  - Here we discuss CSP v1.1 (February 11, 2014)

- default-src
  - Takes a sourcelist as value
  - Default for all resources, unless overridden by specific directives
  - Only allowed resources are loaded

CSP source lists

- Space delimited list of sources
  - ‘self’
  - ‘none’
  - origin(s)
- Examples
  - https://mydomain.com
  - https://mydomain.com:443
  - http://134.58.40.10
  - https://*.mydomain.com
  - https:
  - *://mydomain.com

CSP set of directives (2)

- script-src
  - From which sources, scripts are allowed to be included
- object-src
  - Flash and other plugins
- style-src
  - stylesheets
- img-src
  - images
- media-src
  - sources of video and audio

CSP set of directives (3)

- child-src
  - list of origins allowed to be embedded as frames
  - replaces the deprecated frame-src directive
- font-src
  - web fonts
- connect-src
  - To which origins can you connect (e.g. XHR, websockets)
- frame-options
  - Control framing of the page
- sandbox
  - Trigger sandboxing attribute of embeded iframes

CSP requires sites to "behave"

- Inline scripts and CSS is not allowed
  - All scripts need to be externalized in dedicated JS files
  - All style directives need to be externalized in dedicated style files
  - Clean code separation
- The use of *eval* is not allowed
  - To prevent unsafe string (e.g. user input) to be executed

Example: inline scripts

page.html

```
<script>
  function runMyScript() {
    alert('My alert');
  }
</script>

<a href="#" onClick="runMyScript();">
This link shows an alert!</a>
```

Example: externalized scripts

External JS →

page.html

```
<script src="myscript.js"></script>
<a href="#" id="myLink">This link shows an alert!</a>
```

JavaScript code

Binding to page

myscript.js

```
function runMyScript() {
  alert('My alert');
}
document.addEventListener('DOMContentReady',
function () {
  document.getElementById('myLink')
          .addEventListener('click', runMyScript);
});
```

Insecure relaxations, but be careful!

- To temporary allow inline scripts
  - Content-Security-Policy: script-src 'self' 'unsafe-inline'
- To temporary allow eval
  - Content-Security-Policy: script-src 'self' 'unsafe-inline' 'unsafe-eval'
- To temporary allow inline style directives
  - Content-Security-Policy: style-src 'self' 'unsafe-inline'

Script/style nonces and hashes

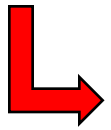
- To allow controlled inline-scripts:
  - Mark your script with a nonce

```
Content-Security-Policy: default-src 'self'; script-src 'self'
https://example.com 'nonce-Nc3n83cnSAd3wc3Sasdfn939hc3‘
```

```
<script nonce="Nc3n83cnSAd3wc3Sasdfn939hc3">
alert("Allowed because nonce is valid.")
</script>
```

  - Add a hash of your inline script to the policy

```
Content-Security-Policy: script-src 'sha256-
YWIzOWNiNzJjNDRlYzc4MTgwMDhmZDIkOWI0NTAyMjgyY2MyMWJl
MWUyNjc1ODJlYWJhNjU5MGU4NmZmNGU3OAo='
```



sha256

```
<script>alert('Hello, world.');</script>
```

DistriNet iMinds KU LEUVEN

CSP reporting feature

- CSP reports violations back to the server owner
  - server owner gets insides in actual attacks
    - i.e. violations against the supplied policy
  - allows to further fine-tune the CSP policy
    - e.g. if the policy is too restrictive
- report-uri directive
  - report-uri /my-csp-reporting-handler
  - URI to which the violation report will be posted

Example violation report

```
Content-Security-Policy: script-src 'self' https://apis.google.com;
report-uri http://example.org/my_amazing_csp_report_parser
```

CSP violation report

```
{
  "csp-report": {
    "document-uri":"http://example.org/page.html",
    "referrer": "http://evil.example.com/",
    "blocked-uri":"http://evil.example.com/evil.js",
    "violated-directive":"script-src 'self' https://apis.google.com",
    "original-policy":"script-src 'self' https://apis.google.com; report-
uri http://example.org/my_amazing_csp_report_parser"
  }
}
```


Based on “HTML5Rocks: An introduction to Content Security Policy” (Mike West)

CSP Reporting: one step further

- Apart from reporting violations via the report-uri directive
- CSP can also run in report only mode
  - Content-Security-Policy-Report-Only: default-src: 'none'; script-src 'self'; report-uri /my-csp-reporting-handler
  - Violation are reported
  - Policies are not enforced

Some CSP examples

- Examples:
  - Mybank.net lockdown
  - SSL only
  - Social media integration
  - Facebook snapshot

Example: mybank.net lockdown

- Scripts, images, stylesheets
  - from a CDN at https://cdn.mybank.net
- XHR requests
  - Interaction with the mybank APIs at https://api.mybank.com
- Iframes
  - From the website itself
- No flash, java, ….

```
Content-Security-Policy: default-src 'none';
script-src https://cdn.mybank.net;
style-src https://cdn.mybank.net;
img-src https://cdn.mybank.net;
connect-src https://api.mybank.com;
child-src 'self'
```


Based on “HTML5Rocks: An introduction to Content Security Policy” (Mike West)

Example: SSL only

- Can we ensure to only include HTTPS content in our website?

```
Content-Security-Policy: default-src https: ;
script-src https: 'unsafe-inline';
style-src https: 'unsafe-inline'
```

- Obviously, this should only be the first step, not the final one!

Example: social media integration

- Google +1 button
  - Script from https://apis.google.com
  - Iframe from https://plusone.google.com
- Facebook
  - Iframe from https://facebook.com
- Twitter tweet button
  - Script from https://platform.twitter.com
  - Iframe from https://platform.twitter.com

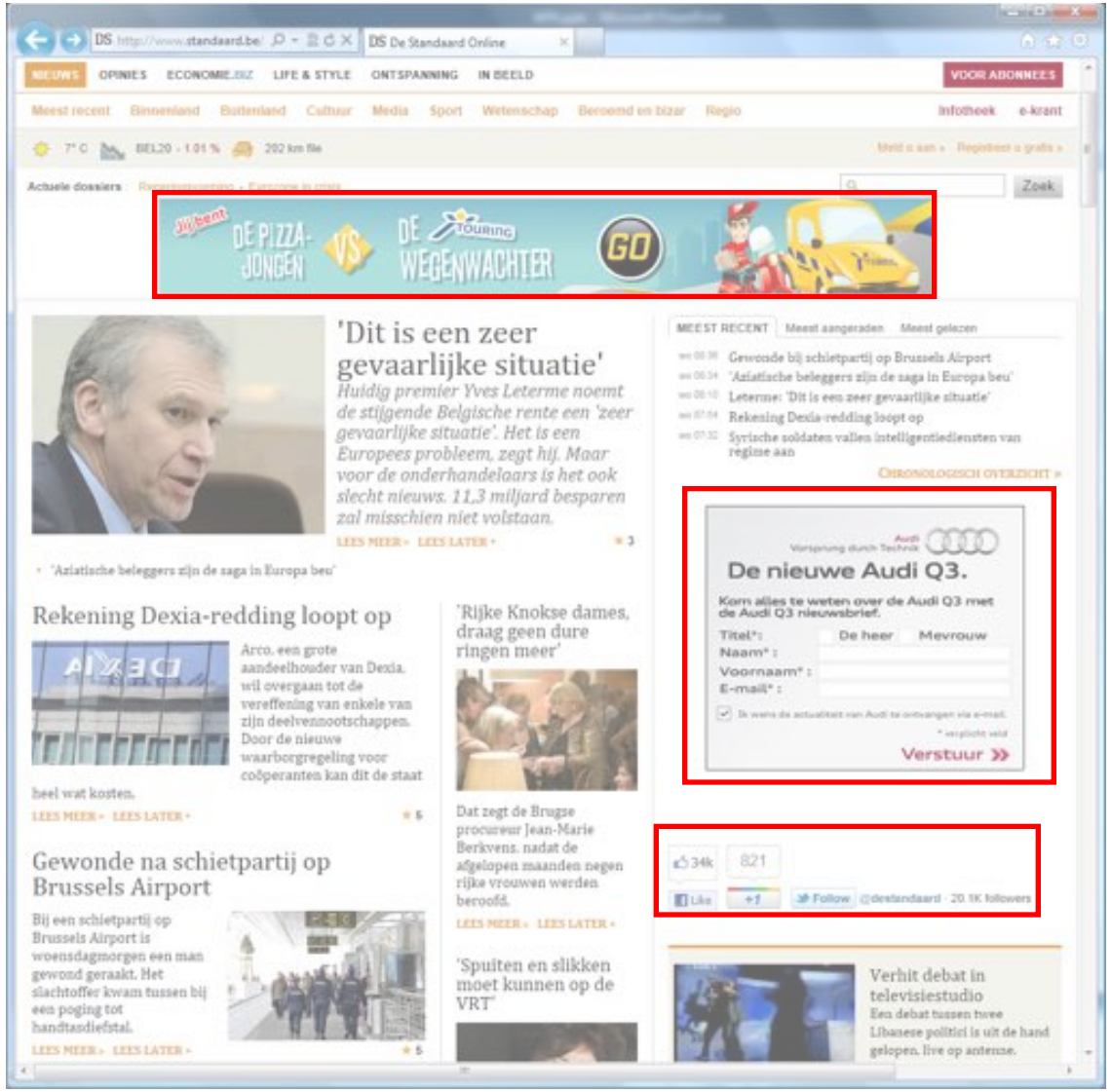
```
Content-Security-Policy:  script-src https://apis.google.com
https://platform.twitter.com;
child-src https://plusone.google.com  https://facebook.com
https://platform.twitter.com
```

Example: Facebook snapshot

```
X-WebKit-CSP: default-src *;
script-src https://*.facebook.com http://*.facebook.com
https://*.fbcdn.net http://*.fbcdn.net *.facebook.net *.google-
analytics.com *.virtualearth.net *.google.com *.spotilocal.com:*
chrome-extension://lifbcibllhkdhoafpjfnlhfpfgnpldfl 'unsafe-inline'
'unsafe-eval' https://*.akamaihd.net http://*.akamaihd.net;style-
src * 'unsafe-inline';
connect-src https://*.facebook.com http://*.facebook.com
https://*.fbcdn.net http://*.fbcdn.net *.facebook.net
*.spotilocal.com:* https://*.akamaihd.net ws://*.facebook.com:*
http://*.akamaihd.net;
```

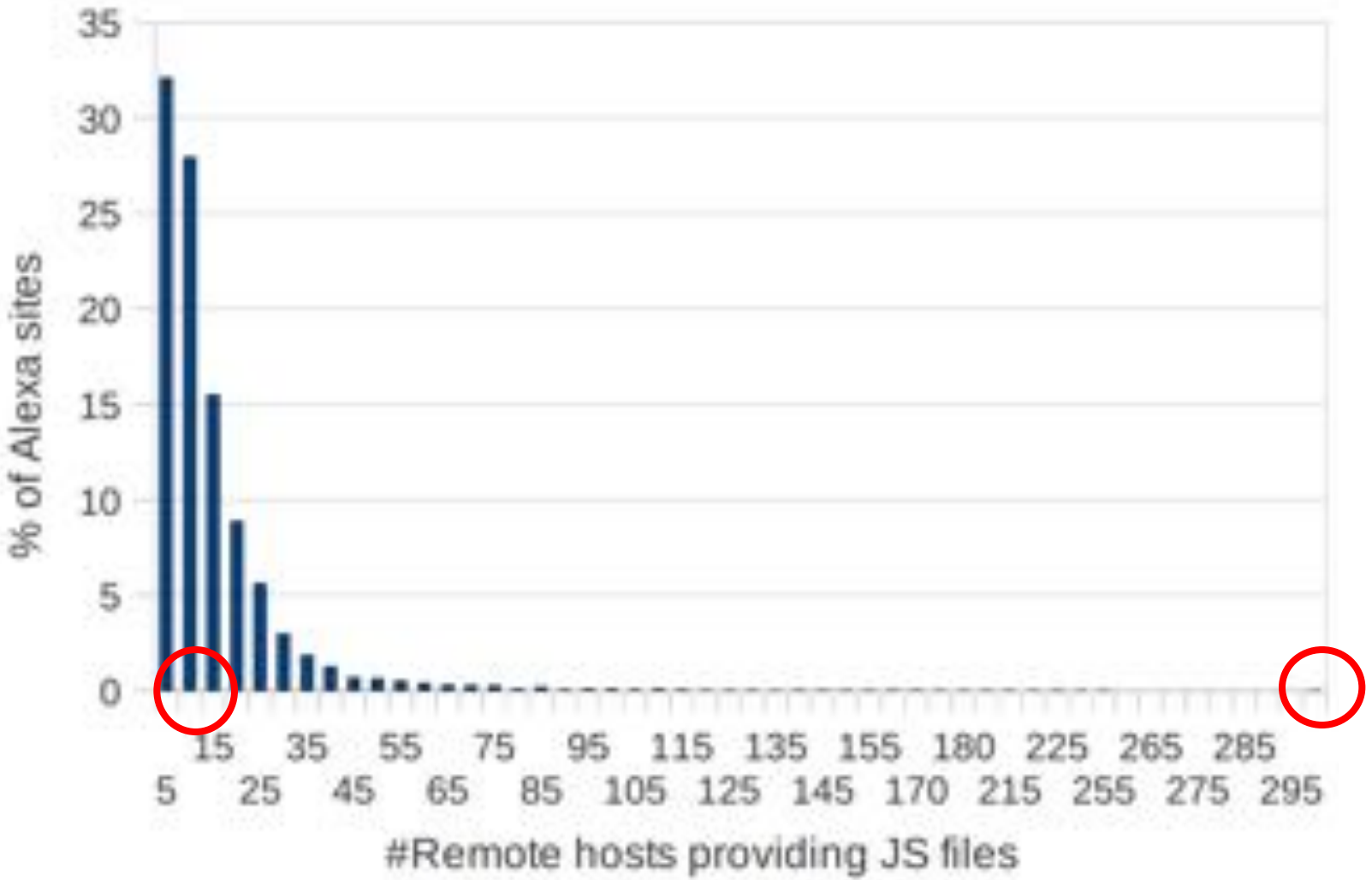
Third-party JavaScript is everywhere

- Advertisements
  - Adhese ad network
- Social web
  - Facebook Connect
  - Google+
  - Twitter
  - Feedsburner
- Tracking
  - Scorecardresearch
- Web Analytics
  - Yahoo! Web Analytics
  - Google Analytics
- …



Number of remote script providers per site

- 88.45% includes at least 1 remote JavaScript library
- 2 out of 3 sites relies on 5 or more script providers
- 1 site includes up to 295 remote script providers



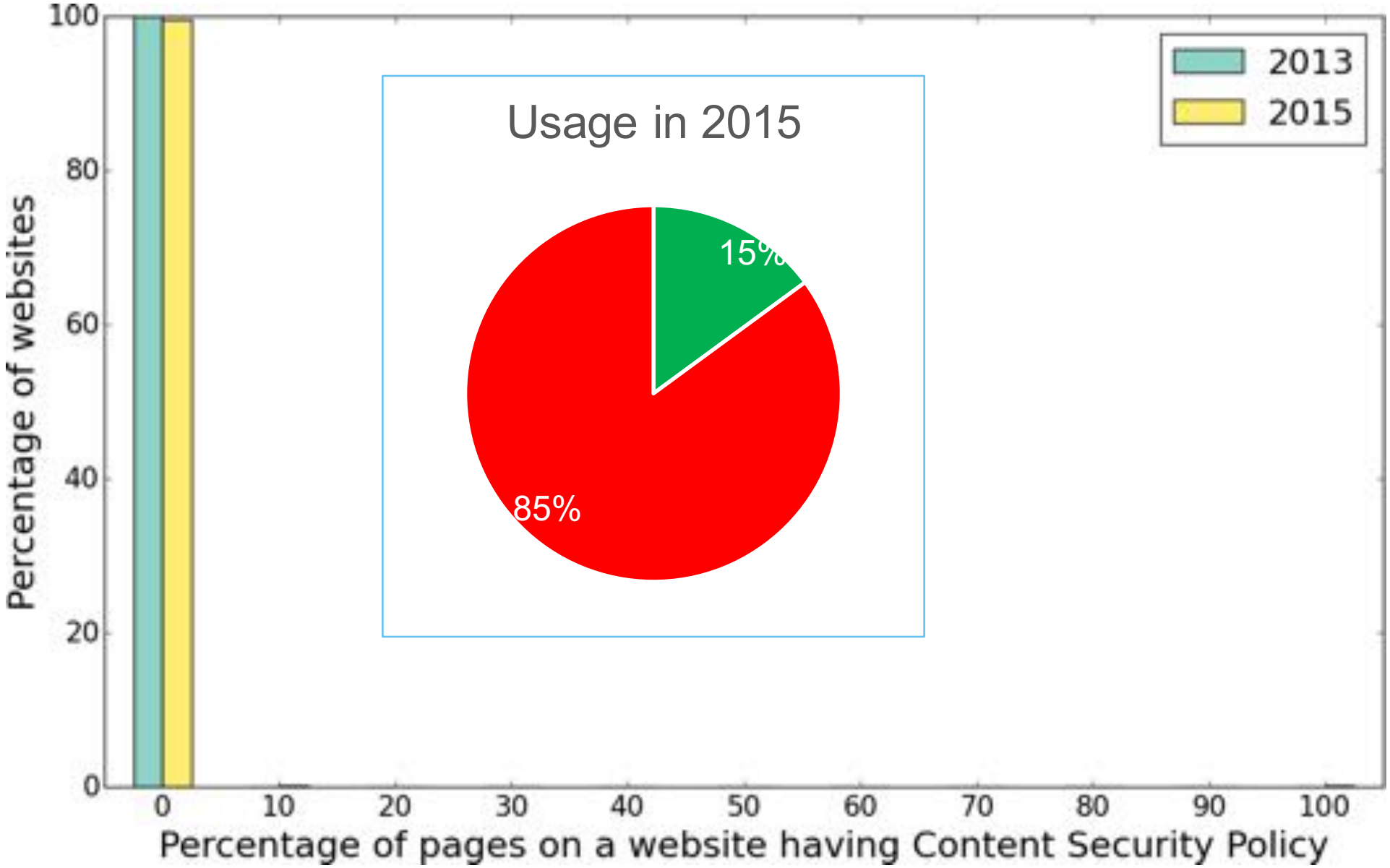
Source: Nick Nikiforakis et. al. You are what you include: Large-scale evaluation of remote JavaScript inclusions. CCS 2012

Most popular JavaScript libraries and APIs

Offered service	JavaScript file	% Alexa Top 10K
Web analytics	www.google-analytics.com/ga.js	68,37%
Dynamic Ads	pagead2.googlesyndication.com/pagead/show_ads.js	23,87%
Web analytics	www.google-analytics.com/urchin.js	17,32%
Social Networking	connect.facebook.net/en_us/all.js	16,82%
Social Networking	platform.twitter.com/widgets.js	13,87%
Social Networking & Web analytics	s7.addthis.com/js/250/addthis_widget.js	12,68%
Web analytics & Tracking	edge.quantserve.com/quant.js	11,98%
Market Research	b.scorecardresearch.com/beacon.js	10,45%
Google Helper Functions	www.google.com/jsapi	10,14%
Web analytics	ssl.google-analytics.com/ga.js	10,12%

Source: Nick Nikiforakis et. al. You are what you include: Large-scale evaluation of remote JavaScript inclusions. CCS 2012

CSP: State of practice



CSP 1.0: State of practice

What's next in CSP?

- CSP 2.0 introduces a set of new directives
- DOM events are now fired upon violations
  - Allows the application to be CSP-aware
- Extensions to CSP:
  - **`upgrade-insecure-requests`** instructs the browser to fetch resources over https
  - **`block-all-mixed-content`** achieves strict mixed content checking

CSP 2.0: State of practice

Subresource Integrity (SRI)

- A lot of resources are served by third-party services (content delivery networks)
  - "Either you trust a CDN, or you host your scripts yourself"
- SRI guarantees the integrity of scripts loaded in the browser

```
<script src="https://code.jquery.com/jquery-2.1.3.min.js"
        integrity="sha256-TXuiaAJuML3…uMLTXuiaAJ3"
        crossorigin="anonymous"></script>
```

Subresource Integrity

- Allows you to specify a hash of an external resource
  - Using the *integrity* attribute on *script* or *link* tags
- Browsers verify this hash before loading the file
  - Refuse to load the file if the hash does not match
- SRI supports the specification of multiple hashes
  - The strongest one available will be used by the browser

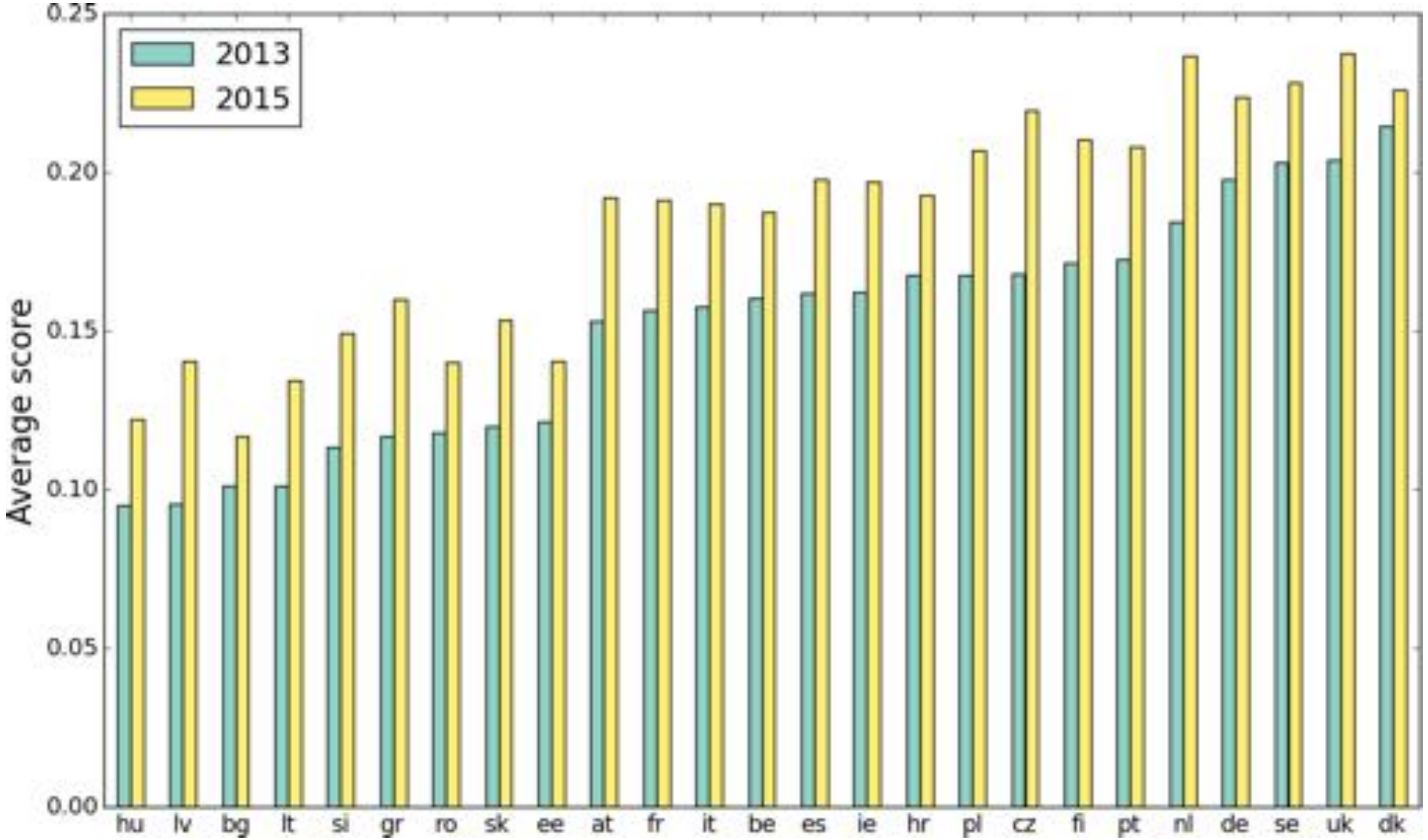
```
<script src="myapplication.js"
  integrity="sha256-… sha512-… ">
</script>
```

```
<link href="myapp.css" type="text/css"
   integrity="sha384-… sha512-…" />
```

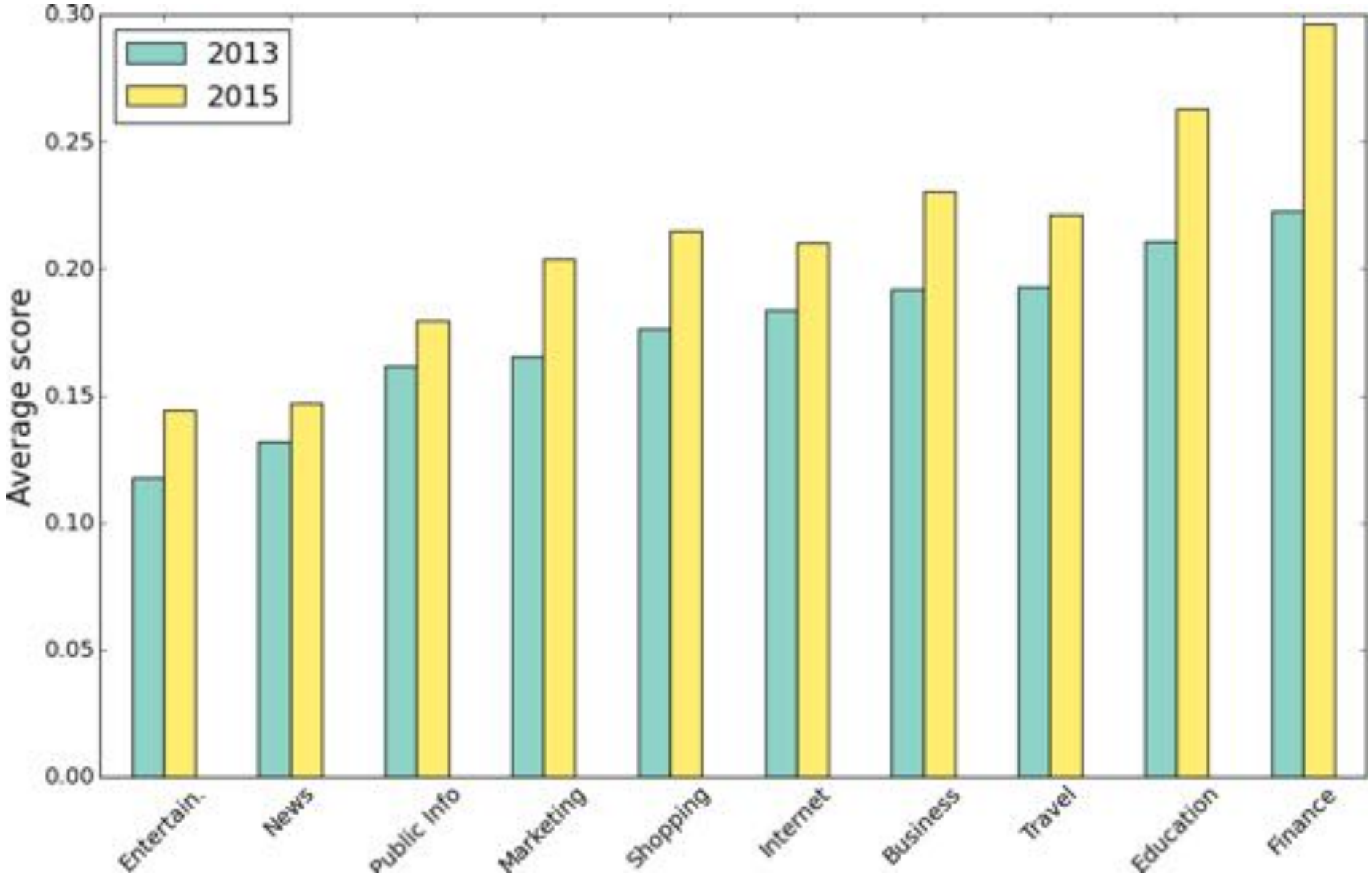
Recap: Mitigating script injection attacks

- HttpOnly flag for session cookies
  - To protect cookies against hijacking and fixation from JavaScript
- X-Content-Type-Options header
  - Disables client-side MIME sniffing
- Content Security Policy (CSP)
  - Domain-level control over resources to be included
  - Most promising infrastructural technique against XSS
  - Interesting reporting-only mode
- SubResource Integrity (SRI)
  - Protects the integrity of third-party served resources

Injection Mitigation Score (1)



Injection Mitigation Score (2)



#3 Framing content securely

Overview

- Attacks:
  - Click-jacking
  - Same domain XSS
- Countermeasures:
  - X-Frame-Options header / frame-ancestors
  - HTML5 sandbox attribute for iframes

Click-jacking

Source: “Busting Frame Busting: a Study of Clickjacking Vulnerabilities on Popular Sites” (W2SP 2010)

Unsafe countermeasures

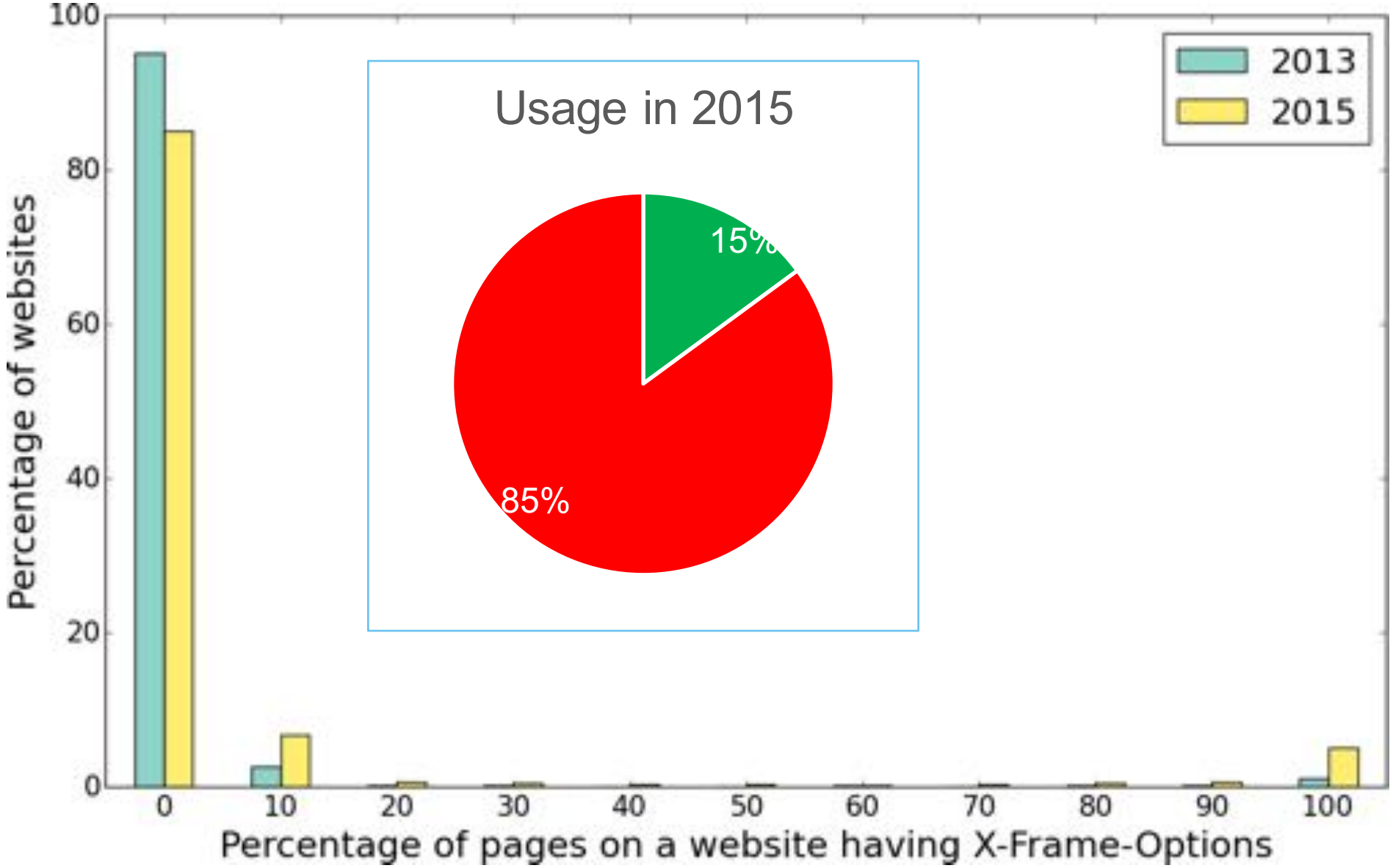
- A lot of unsafe ways exist to protect against clickjacking
  - if (top.location != location)
    top.location = self.location;
  - if (parent.location != self.location)
    parent.location = self.location;
- Can easily be defeated by
  - Script disabling/sandboxing techniques
  - Frame navigation policies
  - XSS filters in browsers

Source: "Busting Frame Busting: a Study of Clickjacking Vulnerabilities on Popular Sites" (W2SP 2010)

X-Frame-Options

- Issued by the HTTP response header
  - X-Frame-Options: SAMEORIGIN
  - Indicates if and by who the page might be framed
- 3 options:
  - DENY
  - SAMEORIGIN
  - ALLOW-FROM uri

XFO: State of practice (deprecated)



XFO has been integrated in CSP

CSP 1.1

- New CSP directive: frame-ancestors
  - Content-Security-Policy: frame-ancestors https://partnerA.com https://partnerB.com
- In contrast to X-Frame-Options, a sourcelist is allowed
  - Common advice is to tailor per partner

Limitations of framing content in same origin

- Iframe integration provides a good isolation mechanism
  - Each origin runs in its own security context, thanks to the Same-Origin Policy
  - Isolation only holds if outer and inner frame belong to a different origin
- Hard to isolate untrusted content within the same origin

HTML5 sandbox attribute

- Expressed as attribute of the iframe tag
  - <iframe src= "/untrusted-path/index.html" sandbox></iframe>
  - <iframe src="/untrusted-path/index.html" sandbox= "allow-scripts"></iframe>

- Level of Protection
  - Coarse-grained sandboxing
  - ‘SOP but within the same domain’

Default sandbox behavior

- Plugins are disabled
- Frame runs in a unique origin
- Scripts can not execute
- Form submission is not allowed
- Top-level context can not be navigated
- Popups are blocked
- No access to raw mouse movements data

Sandbox relaxation directives

- Relaxations:
  - allow-forms
  - allow-popups
  - allow-pointer-lock
  - allow-same-origin
  - allow-scripts
  - allow-top-navigation
- Careful!
  - Combining allow-scripts & allow-same-origin voids the sandbox isolation
- Plugins can not be re-enabled

HTML5 sandbox

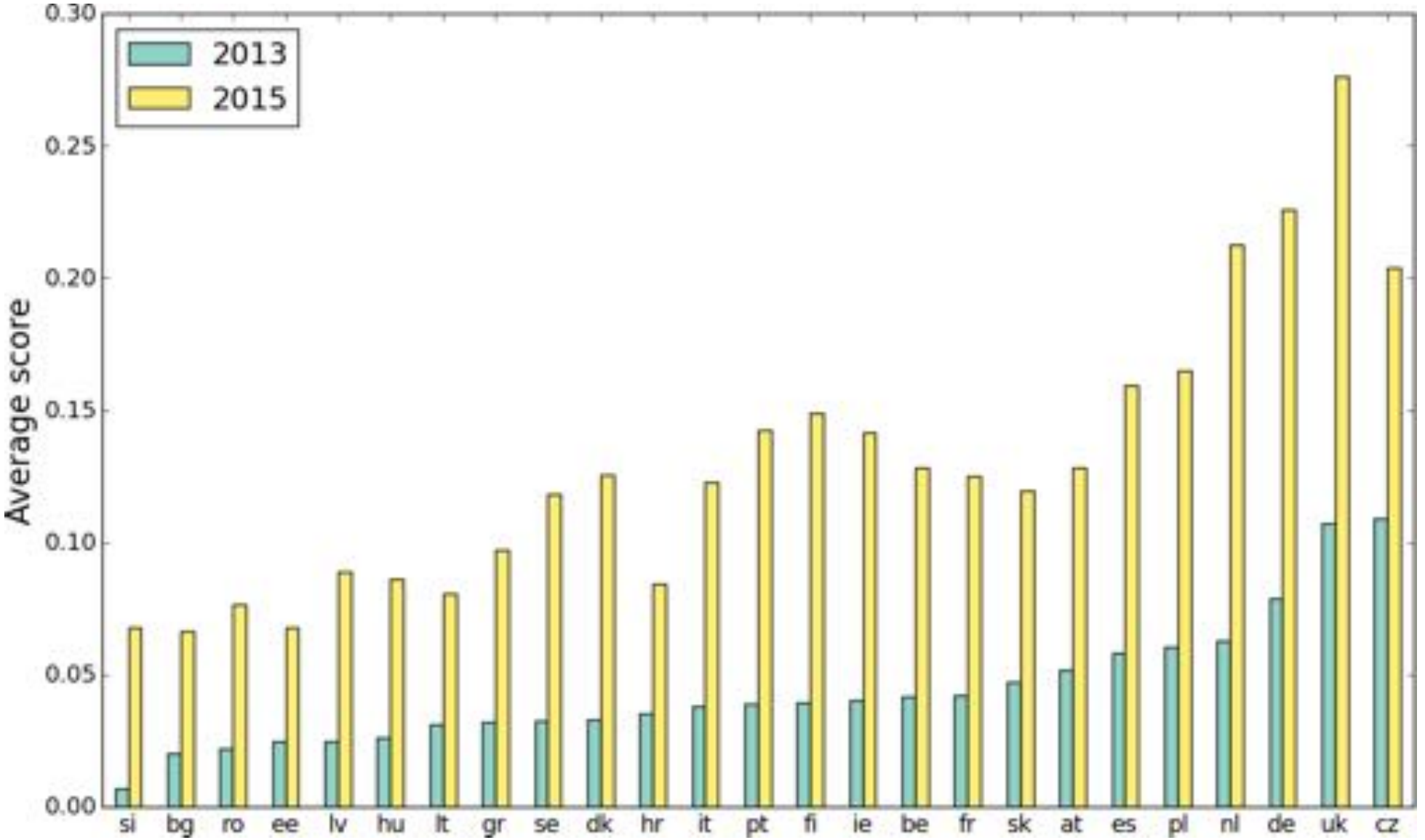
Sandbox has been integrated in CSP

- New CSP directive: sandbox
  - Content-Security-Policy: sandbox
  - Content-Security-Policy: sandbox allow-scripts
- Similar options apply:
  - allow-forms
  - allow-pointer-lock
  - allow-popups
  - allow-same-origin
  - allow-scripts
  - allow-top-navigation

Recap: Framing content securely

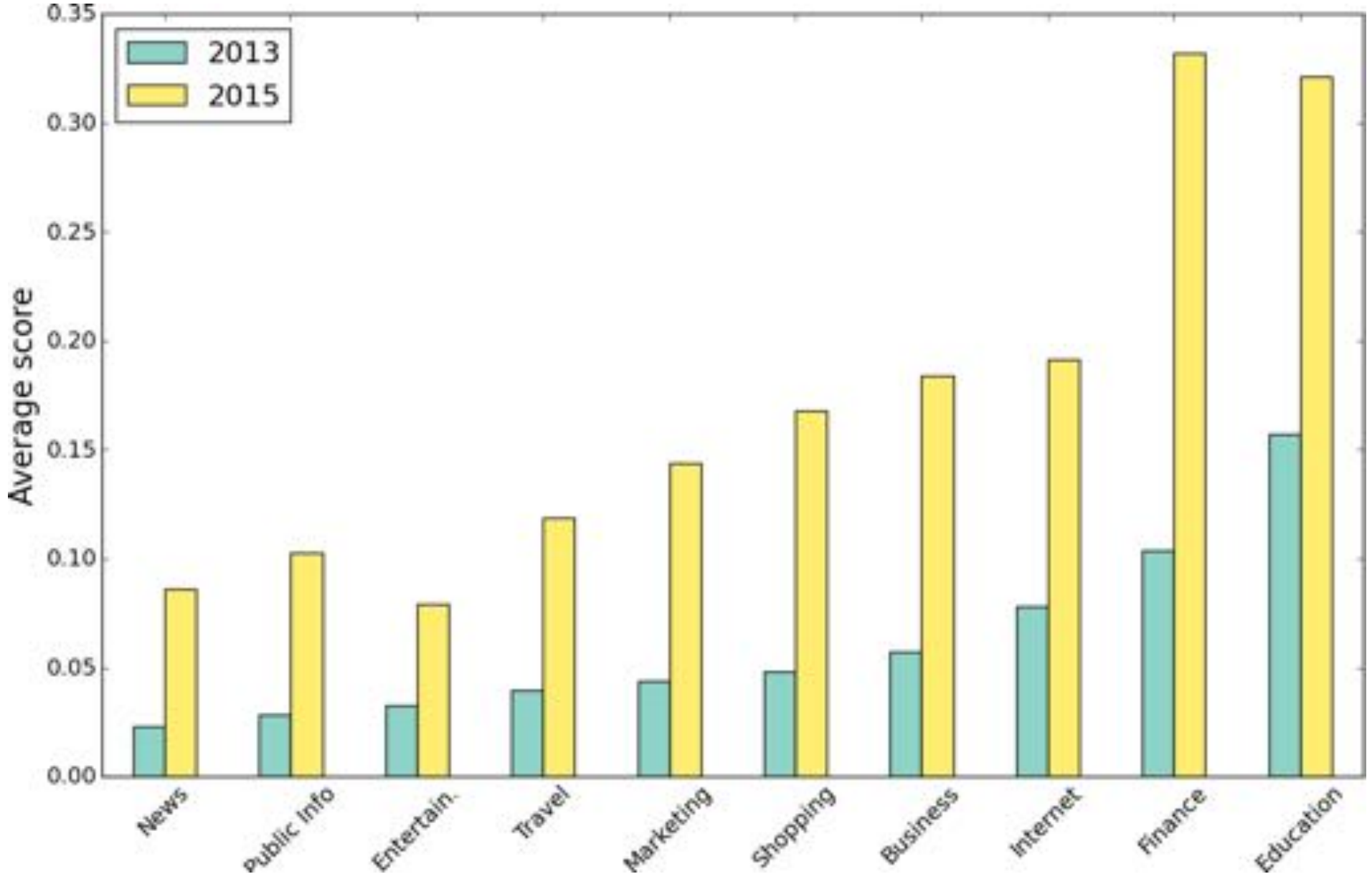
- CSP: Frame ancestors
  - Robust defense against click-jacking
  - Any state-changing page should be protected
- CSP: Sandbox attribute
  - Coarse-grained sandboxing of resources and JavaScript
  - Interesting enabler for security architectures

Secure Framing Score (1)



DistriNet
iMinds
KU LEUVEN

Secure Framing Score (2)



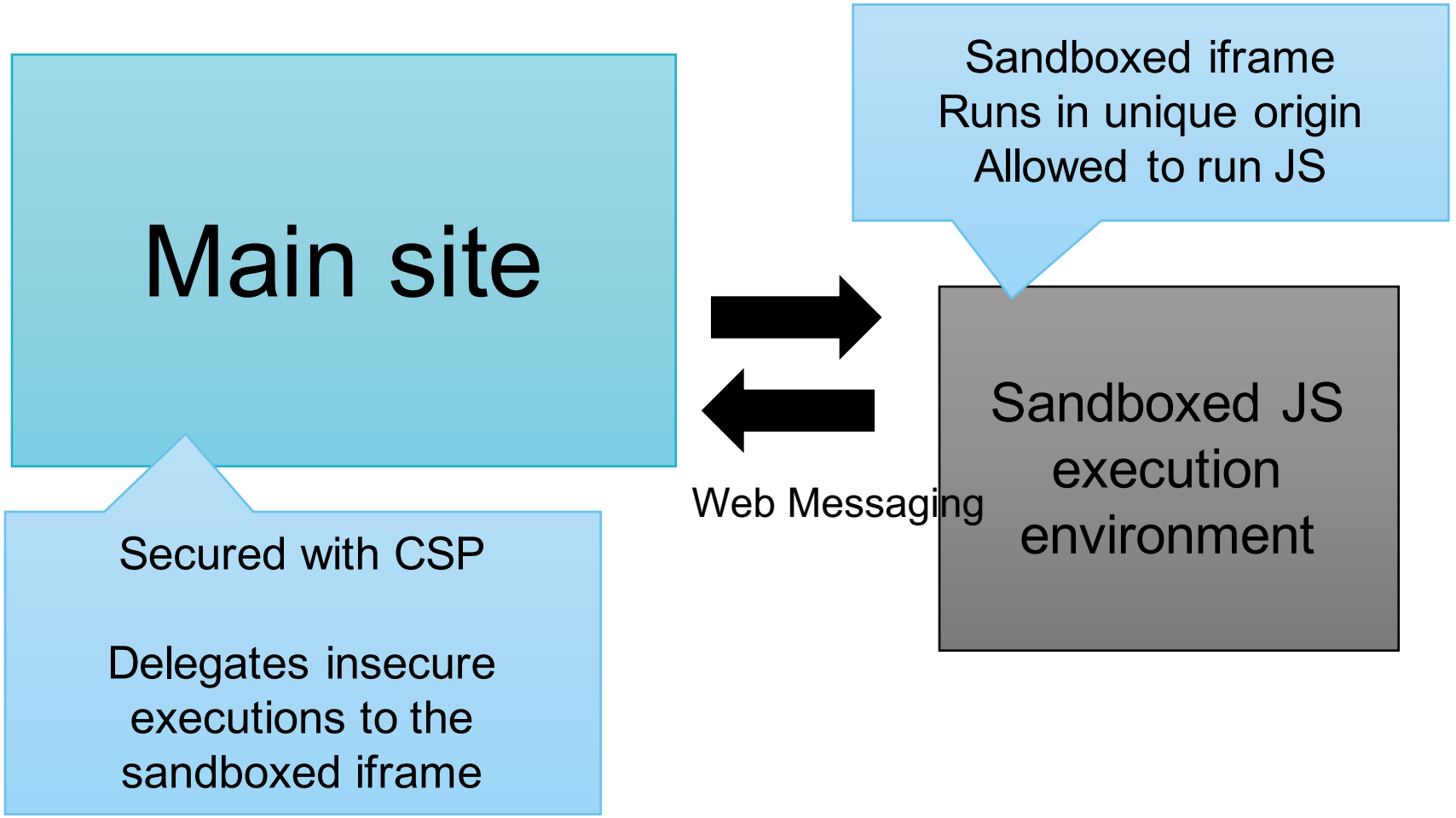
Example security architecture: Combining CSP & Sandbox

"Securing the Client-Side: Building safe web applications with HTML5" (Mike West, Devoxx 2012)

CSP & HTML5 sandbox as security enabler

- Combination of CSP and HTML5 sandbox
  - Enabling technologies for drafting a web application security architecture
  - Allows to define whether or not certain functions/scripts are allowed to run in the origin of the site
- Presented by Mike West at Devoxx 2012
  - Used for document rendering in ChromeOS, …

Example of sandboxing unsafe javascript



Main page (index.html)

Content-Security-Policy: script-src 'self'

```
<html><head>
   <script src="main.js"></script>
</head>
<body>
   <a href="#" id="sandboxFrame"/>Click here</a>
   <iframe id="sandboxFrame" sandbox="allow-scripts"
src="sandbox.html">
   </iframe>
   <div ="#content"></div>
</body></html>
```

Sandboxed frame (sandbox.html)

```
<html><head>
   <script>
          window.EventListener('message', function(event){
           var command = event.data.command;
                 var context = event.data.context;
                 var result = callUnsafeFunction(command, context);
                 event.source.postMessage({
                        html: result}, event.origin);
                 });
   </script>
</head></html>
```

Main script (main.js)

```
document.querySelector('#click').addEventListener('click',
 function(){
   var iframe = document.querySelector('#sandboxFrame');
          var message = {
            command = 'render';
            context = {thing: 'world'}};
          iframe.contentWindow.postMessage(message, '*');
});

window.addEventListener('message', function(event){
 //Would be dangerous without the CSP policy!
 var content = document.querySelector('#content');
 content.innerHTML = event.data.html;
});
```

And what's next?

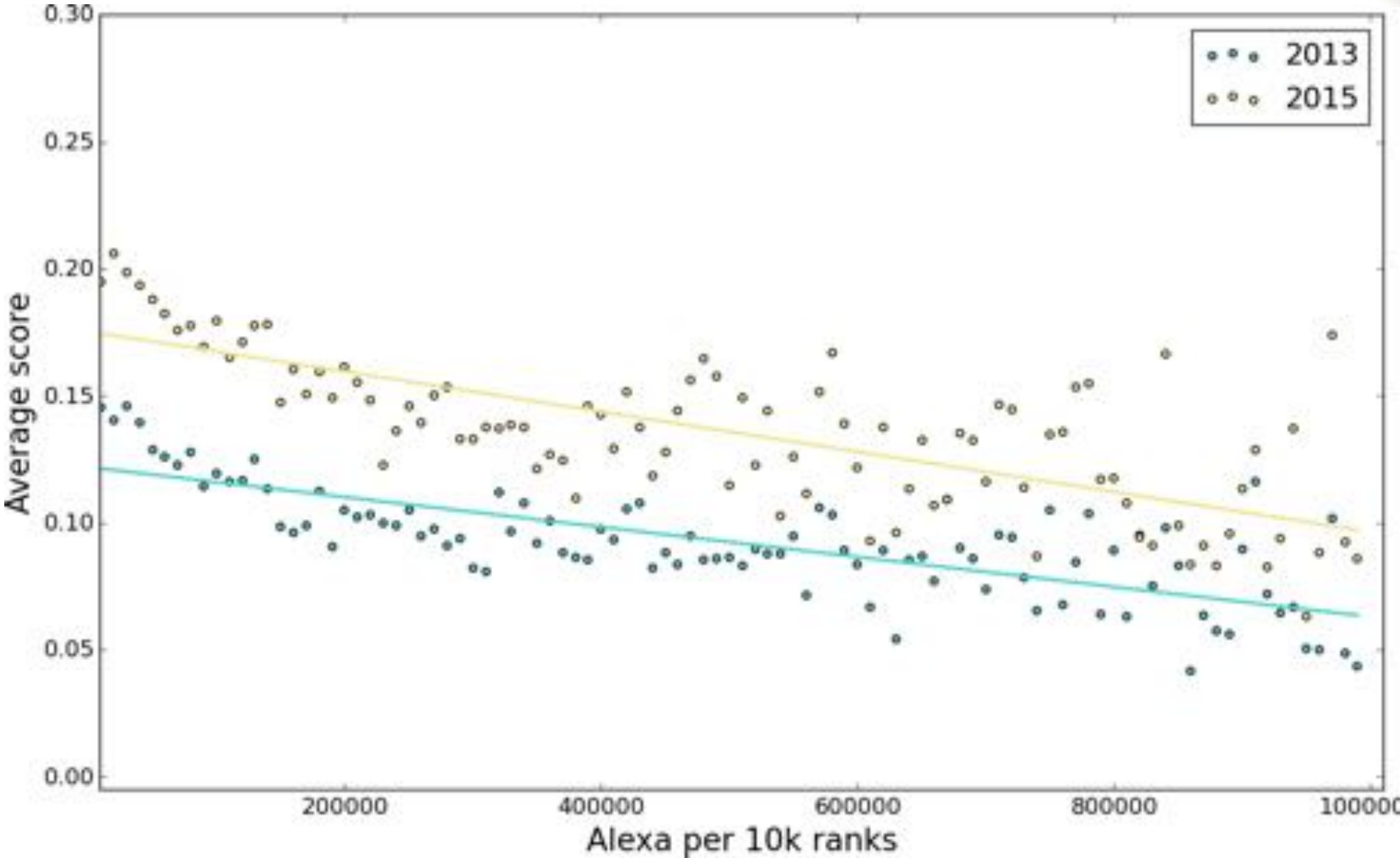
- Seamless integrating unsafe input with the sandbox attribute
  - <iframe **sandbox seamless srcdoc**="<p>Some paragraph</p>"> </iframe>
- seamless attribute
  - Renders visually as part of your site
  - Only for same-origin content
- srcdoc attribute
  - Content as a attribute value instead of a remote page

Wrap-up

Conclusion

- Whole new range of security features
  - Browser-side enforcement, under control of the server
- NOT a replacement of secure coding guidelines, but an interesting additional line of defense for
  - Legacy applications
  - Newly deployed applications
- And most probably, there is many more to come in the next few years…

Is there a correlation between security features and website popularity?

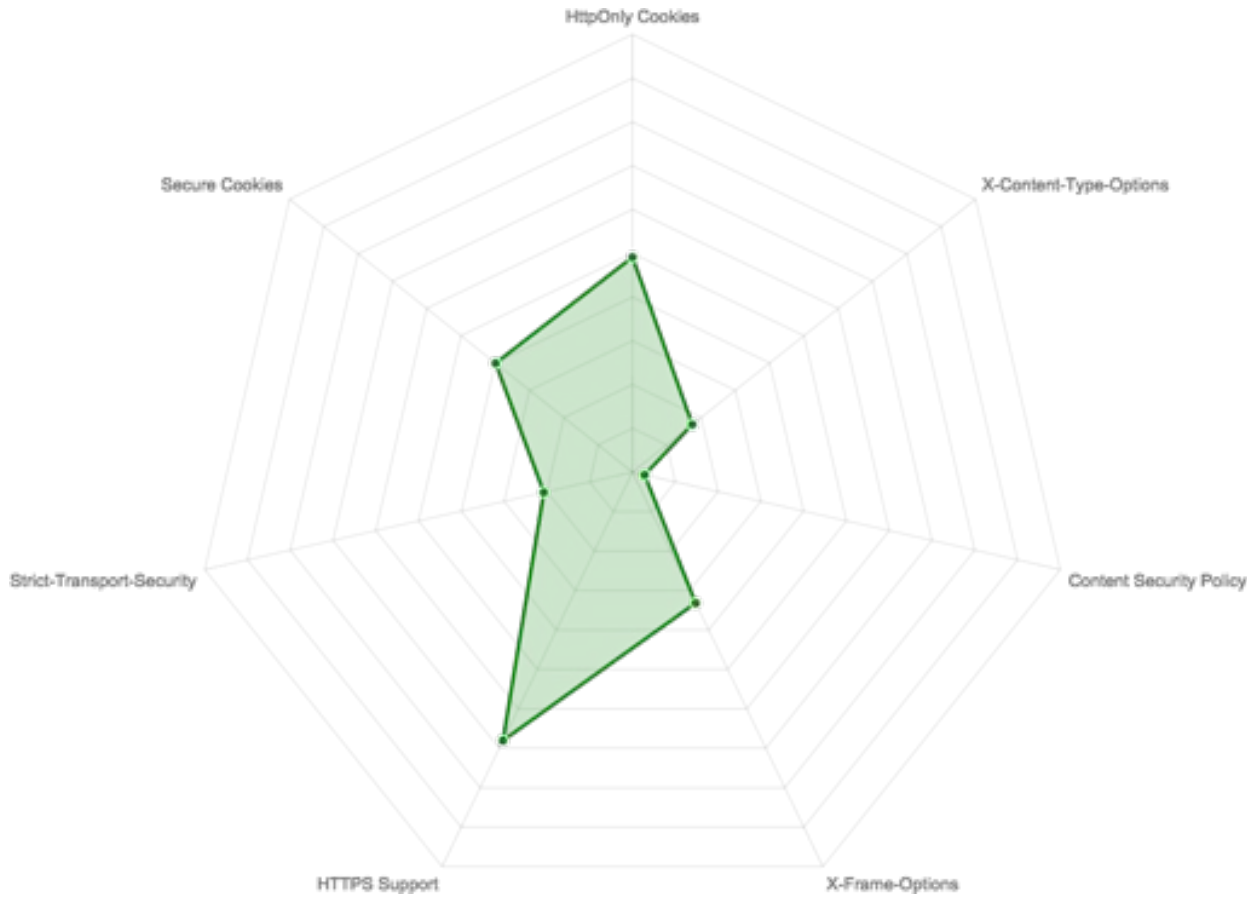


State-of-practice ?

- Secure Communication
  - HTTPS support
  - Secure Cookies
  - Strict-Transport-Security
- XSS Protection
  - HttpOnly cookies
  - X-Content-Type-Options
  - Content-Security-Policy
- Secure Framing
  - X-Frame-Options

Scores per country or vertical ...

Belgium (2015)



Finance (2015)

References

- **STREWS: European Web Security Roadmap, https://www.strews.eu/images/STREWS-D3.2-roadmap.pdf**
- **Primer on Client-Side Web Security, http://www.springer.com/gp/book/9783319122250**
- P. Chen, N. Nikiforakis, L. Desmet and Ch. Huygens. A Dangerous Mix: Large-scale analysis of mixed-content websites (ISC 2013)
- N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. Van Acker, W. Joosen, Ch. Kruegel, F. Piessens, G. Vigna, You are what you include: Large-scale evaluation of remote JavaScript inclusions (CCS 2012)
- Ph. De Ryck et al., Web-platform security guide: Security assessment of the Web ecosystem (STREWS Deliverable D1.1)
- Mike West. An introduction to Content Security Policy (HTML5 Rocks tutorials)
- Mike West. Confound Malicious Middlemen with HTTPS and HTTP Strict Transport Security (HTML5 Rocks tutorials)
- Mike West. Securing the Client-Side: Building safe web applications with HTML5 (Devoxx 2012)

References (2)

- Mike West. Play safely in sandboxed iframes (HTML5 Rocks tutorials)
- Ivan Ristic. Internet SSL Survey 2010 (Black Hat USA 2010)
- Moxie Marlinspike. New Tricks for Defeating SSL in Practice (BlackHat DC 2009)
- B. Sterne, A. Barth. Content Security Policy 1.0 (W3C Candidate Recommendation)
- D. Ross, T. Gondrom. HTTP Header Frame Options (IETF Internet Draft)
- J. Hodges, C. Jackson, A. Barth. HTTP Strict Transport Security (HSTS) (IETF RFC 6797)
- C. Evans, C. Palmer, R. Sleevi. Public Key Pinning Extension for HTTP (IETF Internet Draft)
- Can I use … ?, http://caniuse.com/
- A. Barth, D. Veditz, M. West, Content Security Policy 1.1, W3C Working Draft 11 February 2014
- G.Rydstedt, E. Bursztein, D. Boneh, and C. Jackson. Busting frame busting: a study of clickjacking vulnerabilities at popular sites (W2SP 2010)