Client-side security policies for the web

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

SecAppDev Leuven 2013 (07/03/2013, Leuven)

KU LEUVEN
DISTRINET RESEARCH GROUP
iMinds
CONNECT.INNOVATE.CREATE

About myself

@lieven_desmet

- Lieven Desmet
- Research manager of the iMinds-DistriNet Research Group (KU Leuven, Belgium)
- Active participation in OWASP:
  - → Board member of the OWASP Belgium Chapter
  - → Co-organizer of the academic track on past OWASP AppSec Europe Conferences

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

http://distrinet.cs.kuleuven.be

Web Application Security Team

- Web Session management
  - Session hijacking, fixation, SSL stripping, CSRF,...
  - CSRF protection: CsFire
    - 50K downloads
    - Available for Firefox and Chrome
- Web Mashup Security
  - Secure integration of 3rd party JavaScript
  - Information Flow Control for JavaScript
- Various Web Security Assessments
  - HTML5 security analysis for ENISA
  - Large scale assessments of security state-of-practise

Client-side security policies for the web

KU LEUVEN
DISTRINET RESEARCH GROUP
iMinds
CONNECT.INNOVATE.CREATE

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		
[8]	75.0		
[9]	74.0		
[10]	73.8		
[11]	73.1		
[12]	70.1		
[13]	69.3		
[14]	68.5		
[15]	67.8		
[16]	66.0		
[17]	65.5		
[18]	64.6		
[19]	64.1		
[20]	62.4		
[21]	61.5		
[22]	61.1		
[23]	61.0	CWE-134	Uncontrolled Format String
[24]	60.3	CWE-190	Integer Overflow or Wraparound
[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

ment

A6 – Security Misconfiguration (NEW)

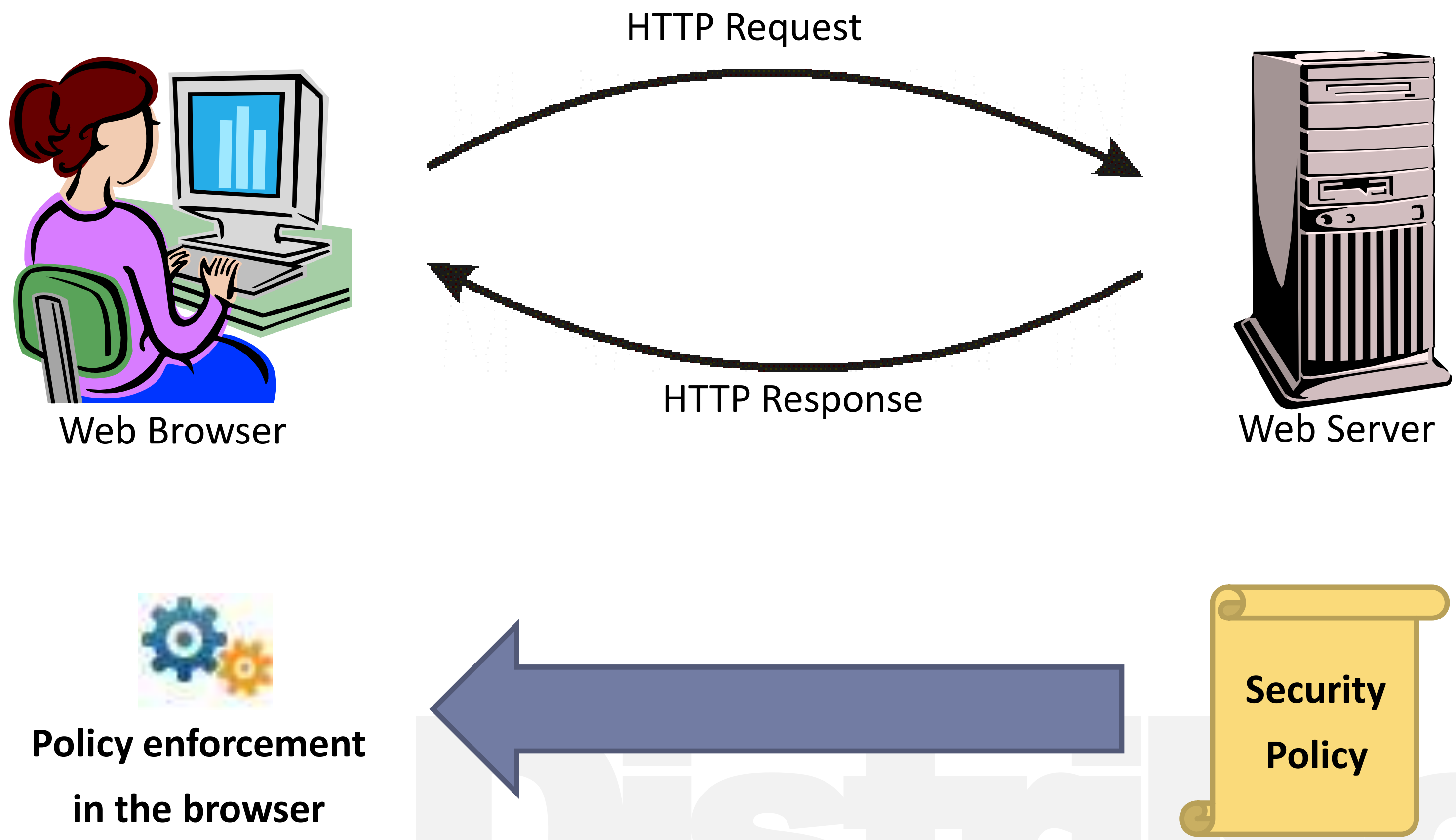
A7 – Insecure Cryptographic Storage

A8 – Failure to Restrict URL Access

A9 – Insufficient Transport Layer Protection

A10 – Unvalidated Redirects and Forwards (NEW)

Client-side security policies in the web



Overview

- Introduction
- Securing browser-server communication
- Mitigating script injection attacks
- Framing content securely
- Enabling cross-domain interactions
- Wrap-up

Introduction

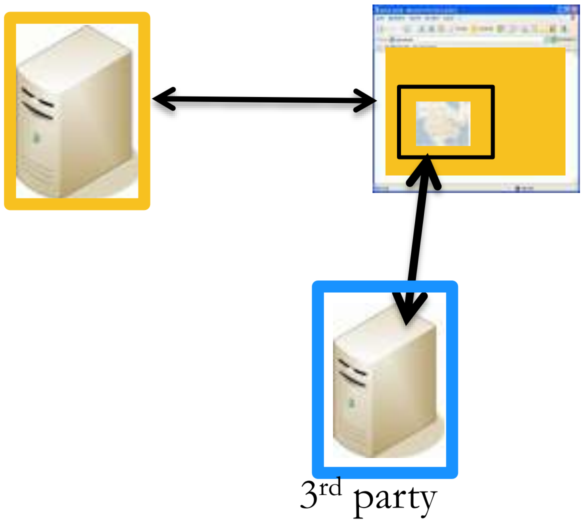
Overview

- 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

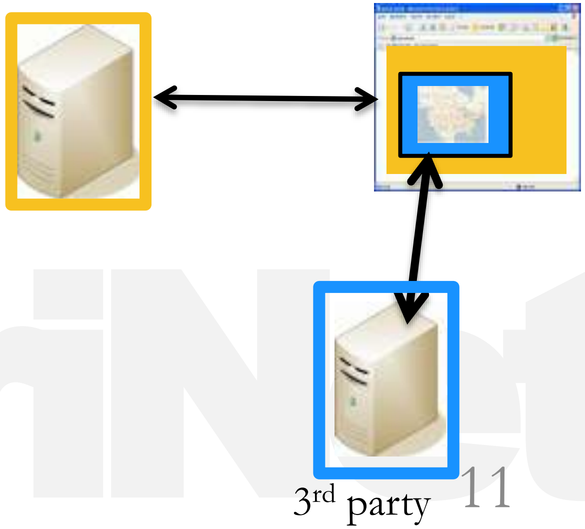
Script inclusion

```
<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>
```

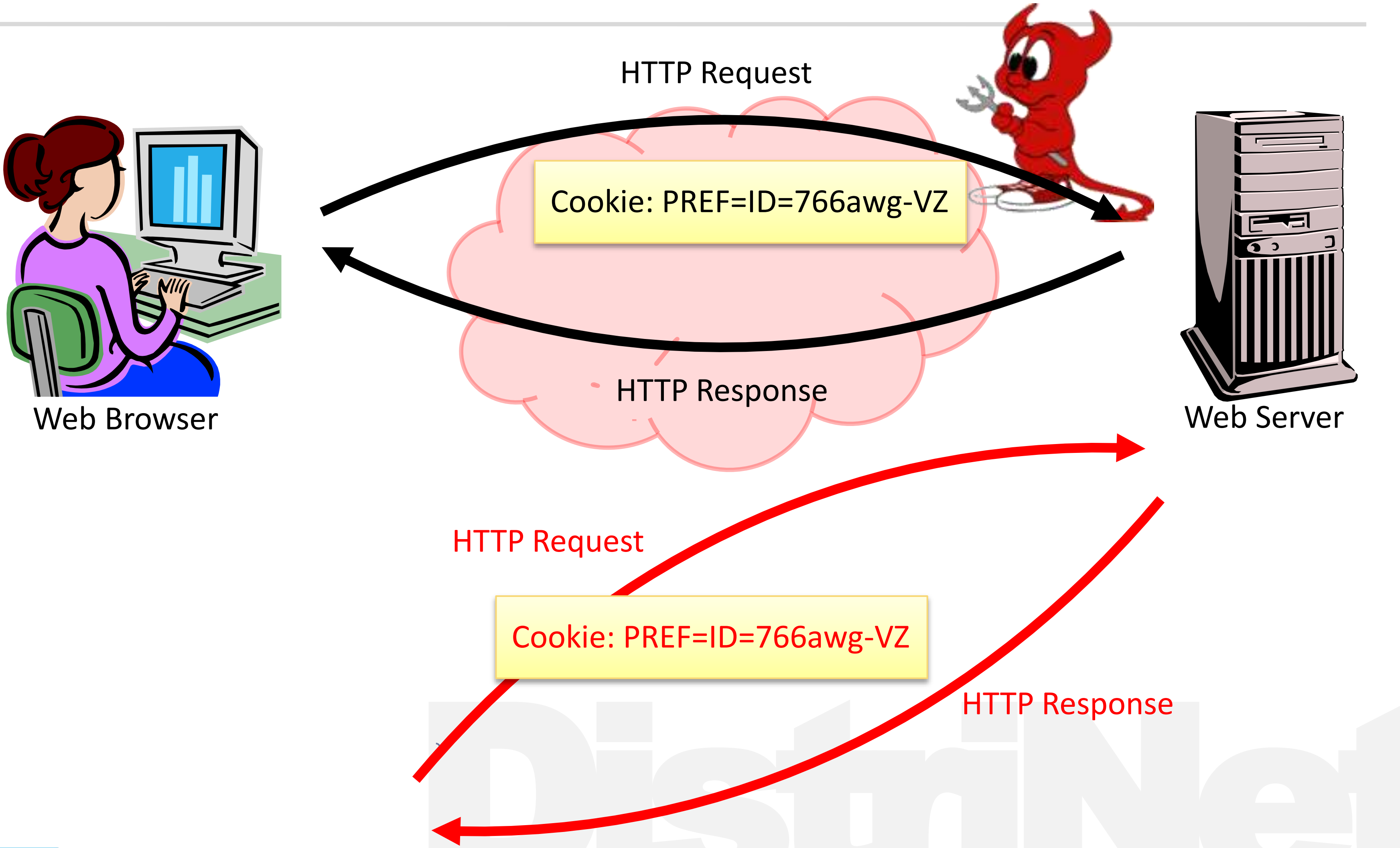


Securing browser-server communication

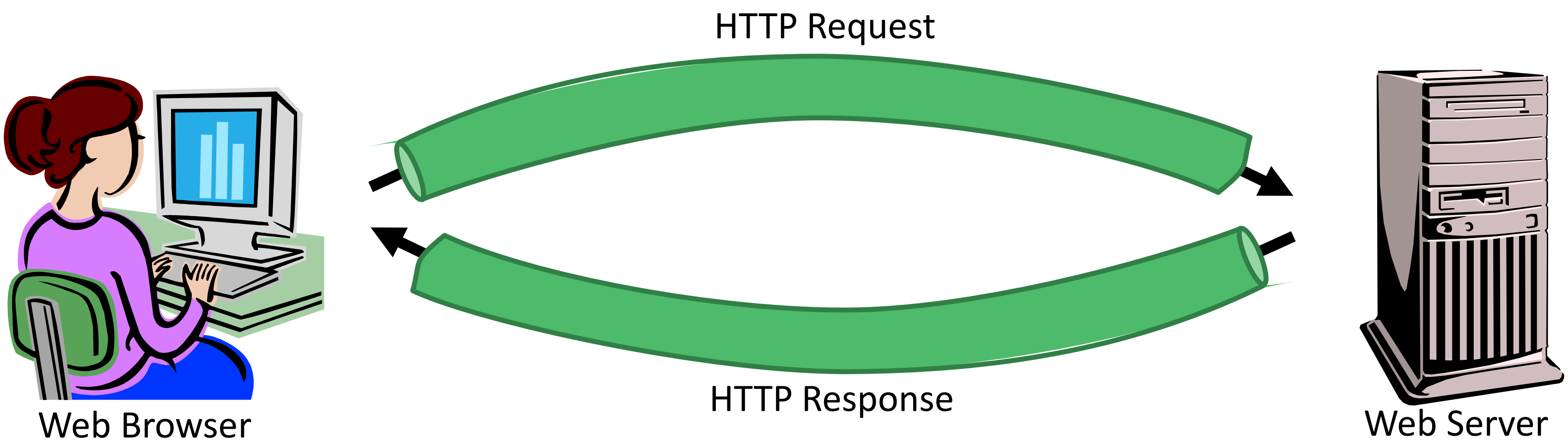
Overview

- 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…



Problem cured?

- TLS usage statistics:
  - → 0.78% of active domains use TLS (with valid SSL certificate)
  - → For Alexa top 1 million: 27.86% use TLS

Internet SSL Survey 2010, Qualys

- Remaining problems:
  - → Mixed use of HTTPS/HTTP and session cookies
  - → SSL Stripping attacks

Mixed use of HTTPS/HTTP

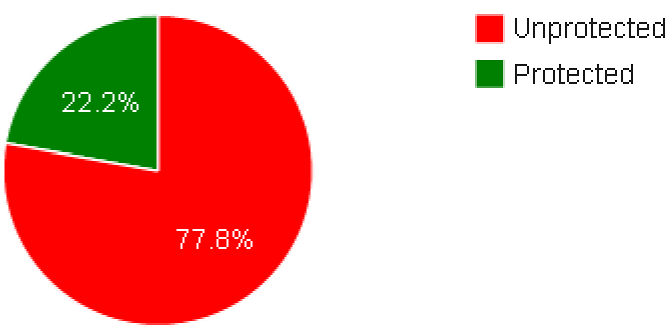
- Cookies are bound to domains, not origins
- By default, cookies are sent both over HTTPS and HTTP
- Any request to your domain over HTTP leaks the (session) cookies…

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 flag: state-of-practice

- Browser compatibility
  - → All recent browsers support the secure flag for cookies
- Usage statistics

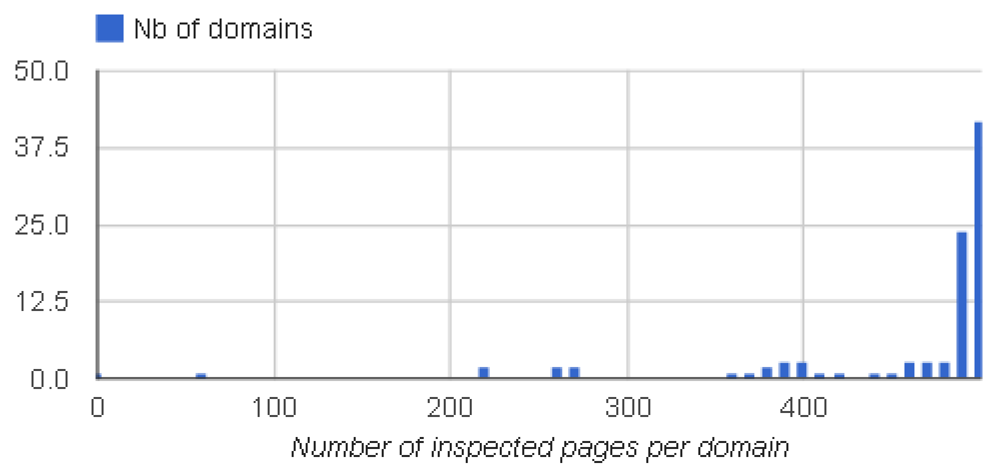


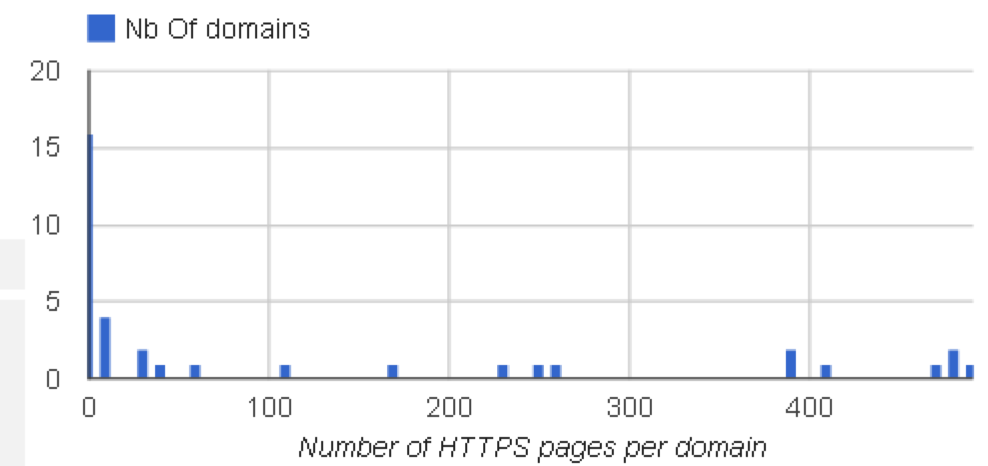
	Domain
1	google.com
2	live.com
3	amazon.com
4	microsoft.com
5	paypal.com
6	bnpparibasfortis.be
7	t.co
8	dropbox.com

Own experiment on top 100 of websites, visited from Belgium (Alexa)

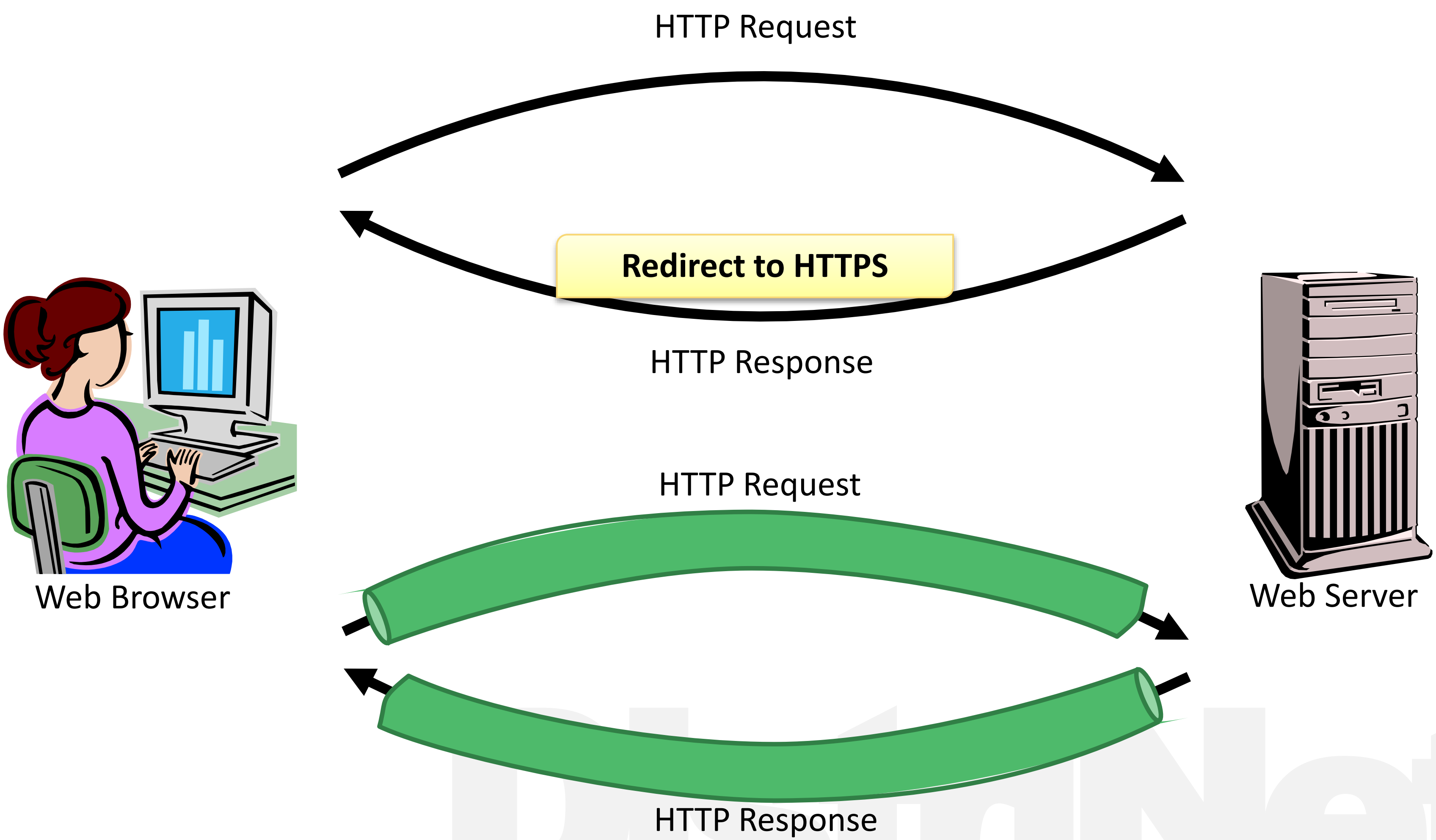
Some background on this experiment

- Number of inspected domains: 96
- Total number of inspected pages: 44431
- Average number of pages per domains: 462
- 36 out 96 domains serve HTTPS pages





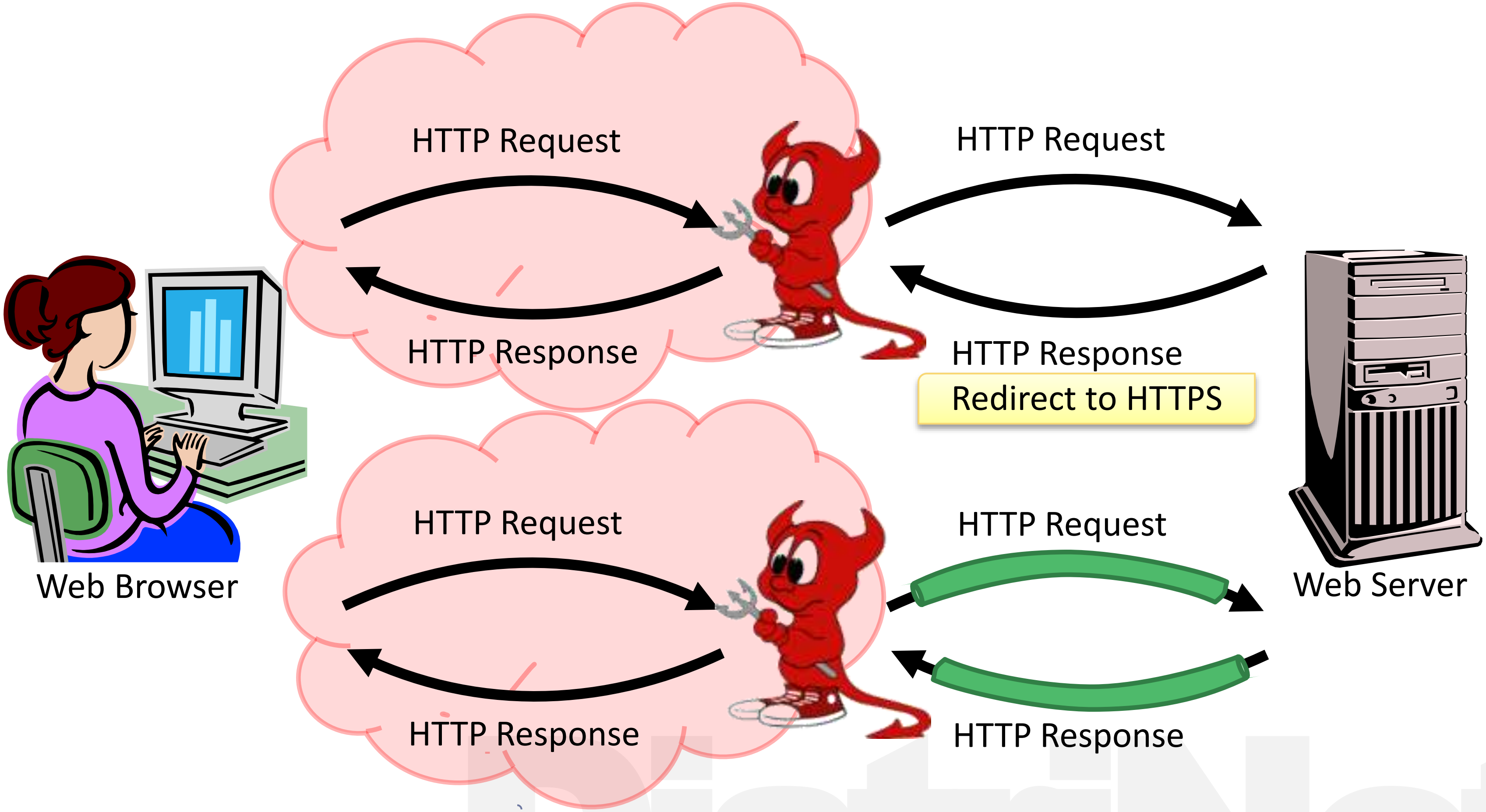
HTTP to HTTPS bootstrapping



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



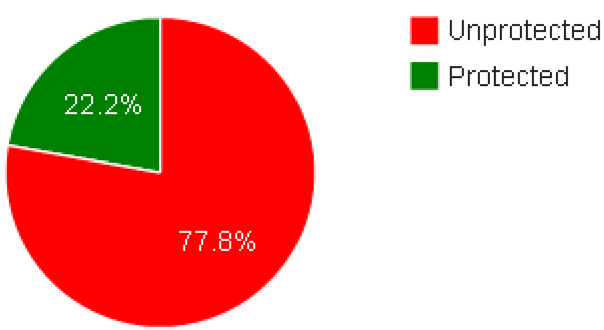
Moxie Marlinspike, BlackHat DC 2009

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: state-of-practice

- Browser compatibility
  - Chrome 4+, Firefox 4+, Opera 12+
- Usage statistics



	Domain	# of HTTPS pages using HSTS	# of HTTPS pages visited	Percentage of pages
1	twitter.com	500	500	100
2	dropbox.com	283	404	70
3	googleusercontent.com	34	443	7
4	paypal.com	34	399	8
5	t.co	19	495	3
6	google.fr	3	500	0
7	google.com	1	499	0
8	google.nl	1	500	0

Own experiment on top 100 of websites, visited from Belgium (Alexa)

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

- Issued as HTTP response header
  - → Public-Key-Pins: max-age=500; pin-sha1="4n972HfV354KP560yw4uqe/baXc="; pin-sha1="IvGeLsbqzPxdI0b0wuj2xVTdXgc="
- Freezes the certificate by pushing a fingerprint of (parts of) the certificate chain to the browser
- Currently an IETF Internet-Draft
- Supported in Chrome 18+

Recap: Securing browser-server communication

- Use of TLS
- 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

Mitigating script injection attacks

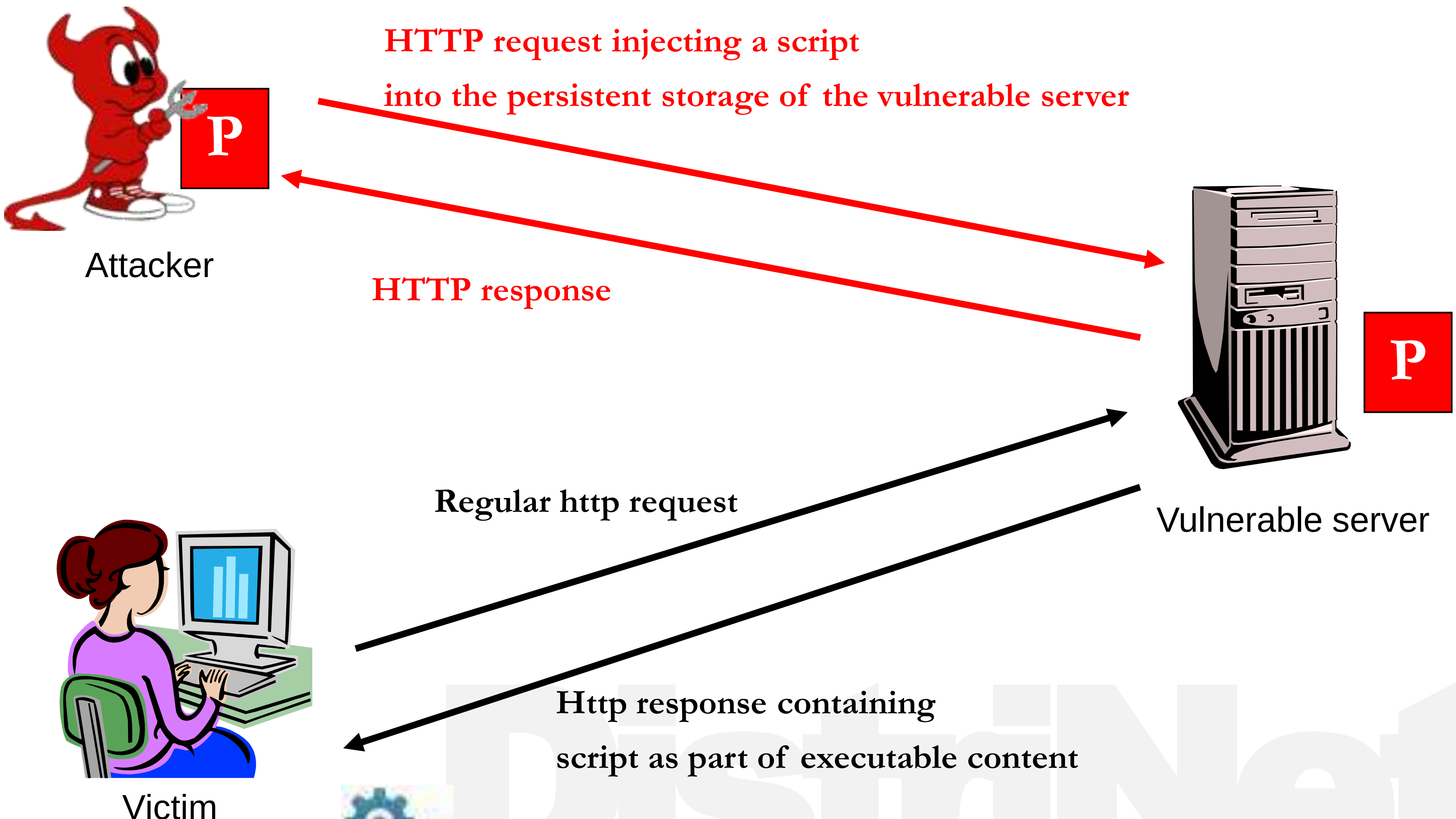





Overview

- Attack:
  - → Cross-Site Scripting (XSS)
- Countermeasures:
  - → HttpOnly flag for session cookies
  - → X-XSS-Protection header
  - → Content Security Policy (CSP)

Example: Stored or persistent XSS



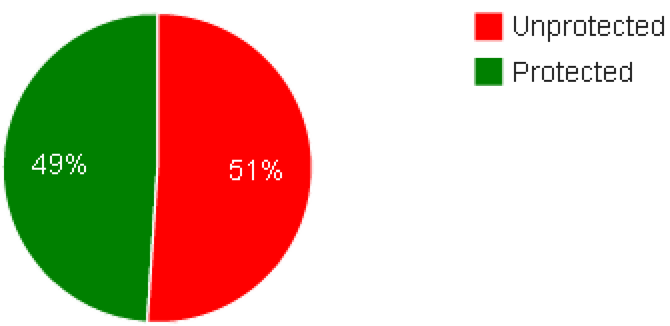
KU LEUVEN
DISTRINET RESEARCH GROUP
iMinds
CONNECT.INNOVATE.CREATE

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

- Browser compatibility
  - → Support in all browsers
  - → Only recently on Android
- Usage statistics



	Domain
1	google.be
2	google.com
3	live.com
4	yahoo.com
5	twitter.com
6	nieuwsblad.be
7	ebay.be
8	google.fr
9	msn.com
10	immoweb.be
11	dhnet.be
12	lesoir.be
13	microsoft.com
14	pinterest.com
15	tumblr.com

Own experiment on top 100 of websites, visited from Belgium (Alexa)

X-XSS-Protection

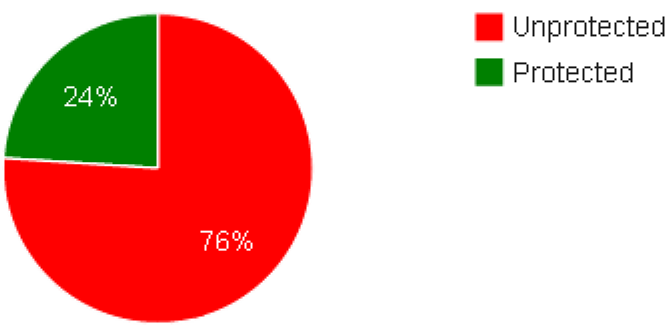
- Best-effort protection in the browser against reflected XSS
  - Can be controlled via the X-XSS-Protection header in the HTTP response
  - On by default
- Completeness of protection
  - Protects only against reflected XSS
  - Multiple bypasses have been reported

X-XSS-Protection: modes of operation

- Default protection
  - → X-XSS-Protection: 1
- Optional opt-out
  - → X-XSS-Protection: 0
- Blocking mode
  - → X-XSS-Protection: 1; mode=block
  - → Prevents the page from rendering

X-XSS-Protection: state-of-practice

- Browser compatibility:
  - → Internet Explorer 8+, Chrome and Safari
- Usage statistics



	Domain	# of pages using x_xss_protection	# of pages visited	Percentage of pages
1	facebook.com	500	500	100
2	google.fr	500	500	100
3	google.nl	500	500	100
4	twitter.com	500	500	100
5	google.be	498	500	99
6	google.com	497	499	99
7	googleusercontent.com	439	443	99
8	live.com	378	463	81
9	t.co	225	495	45
10	blogger.com	223	223	100
11	adcash.com	172	383	44
12	yahoo.com	11	496	2
13	belgium.be	10	500	2
14	ebay.be	10	500	2
15	fgov.be	8	497	1

Own experiment on top 100 of websites, visited from Belgium (Alexa)

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 v1.0
- 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)

- frame-src
  - list of origins allowed to be embedded as frames
- font-src
  - web fonts
- connect-src
  - To which origins can you connect (e.g. XHR, websockets)
- sandbox
  - Optional
  - Trigger sandboxing attribute of included 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

page.html

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

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'

Be careful!

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

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

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;
  frame-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!

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

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;
  frame-src https://plusone.google.com https://facebook.com https://platform.twitter.com

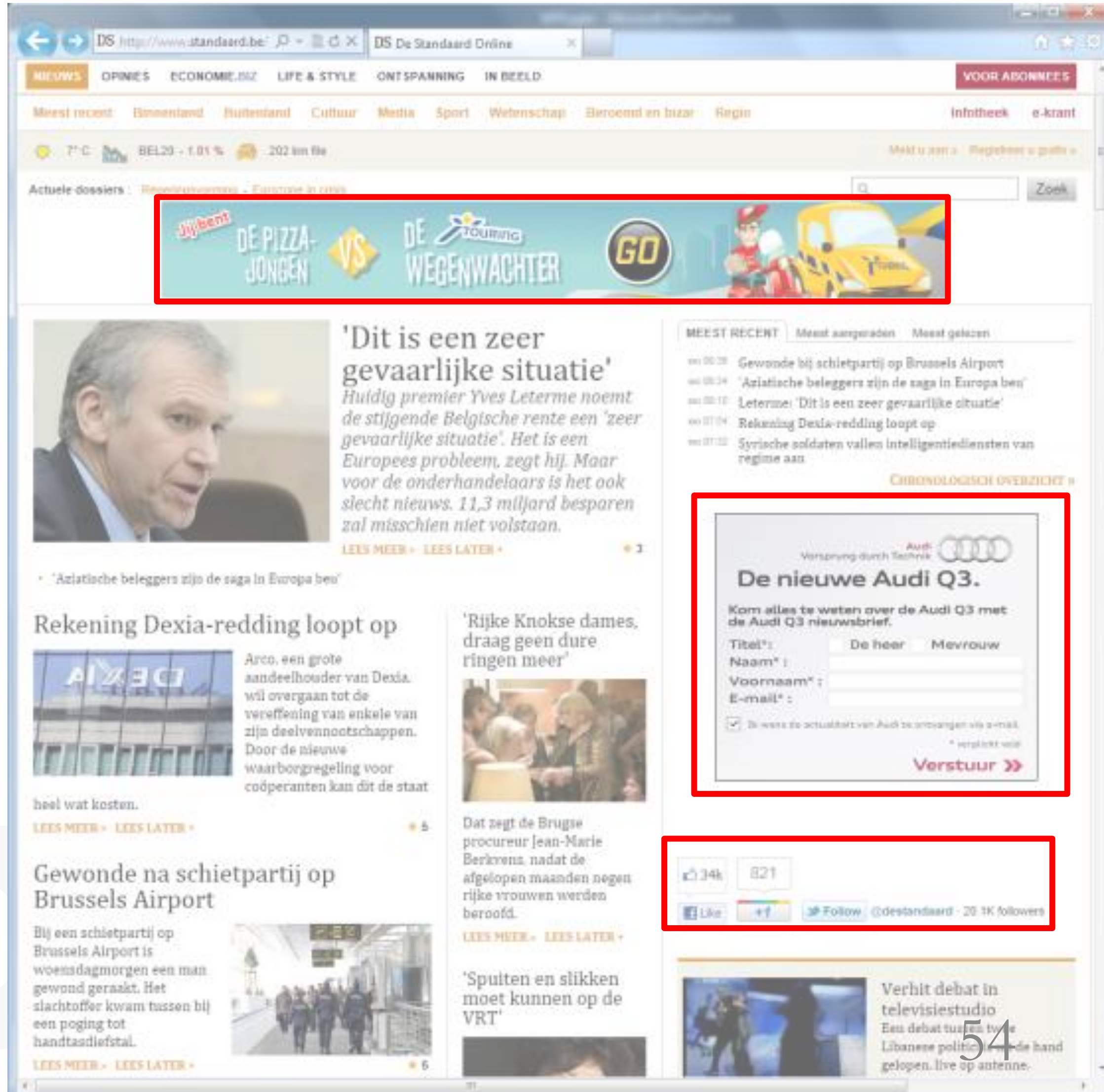
Based on "HTML5Rocks: An introduction to Content Security Policy" (Mike West)

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
- …



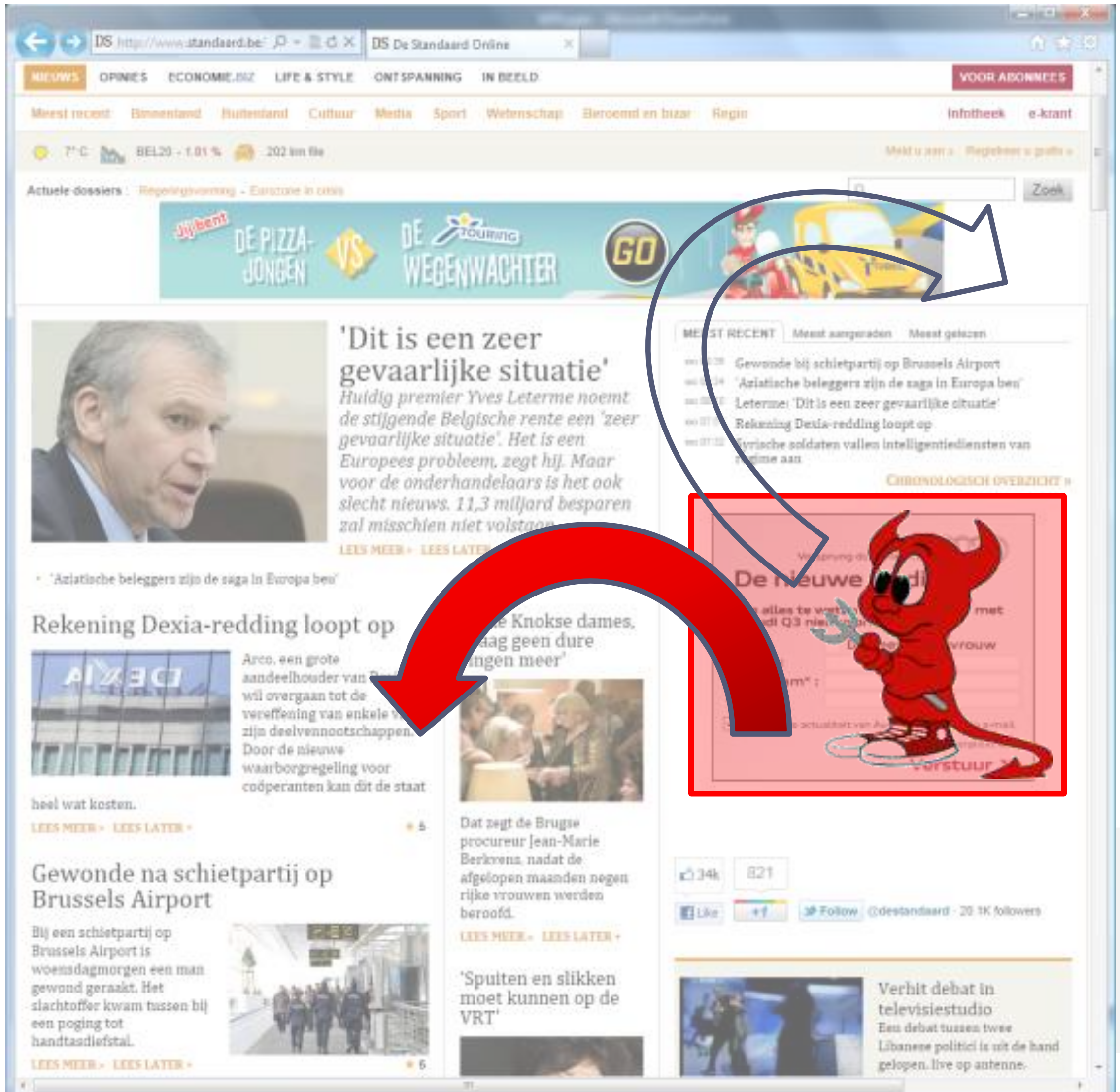
KU LEUVEN DISTRINET RESEARCH GROUP
iMinds CONNECT.INNOVATE.CREATE

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

CCS 2012

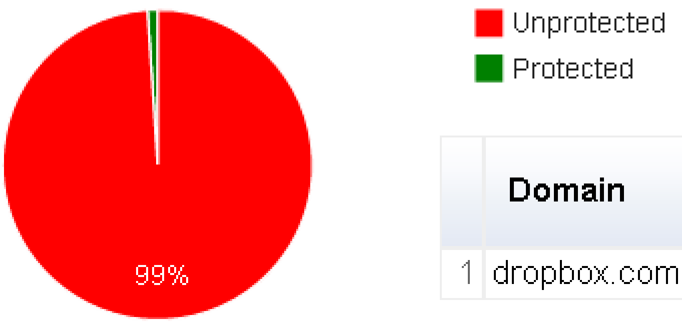
Full experiment and interesting attacks are discussed in detail by the authors in the "Sandboxing JavaScript" sessions

Malicious third-party scripts can …



CSP: state-of-practice

- Browser compatibility:
  - → Firefox and Chrome
  - → Older header names: X-WebKit-CSP, X-Content-Security-Policy
-



	Domain	# of pages using x_content_security_policy	# of pages visited	Percentage of pages
1	dropbox.com	53	404	13

Own experiment on top 100 of websites, visited from Belgium (Alexa)

Recap: Mitigating script injection attacks

- HttpOnly flag for session cookies
  - → To protect cookies against hijacking and fixation from JavaScript
- X-XSS-Protection header
  - → Coarse-grained control over built-in browser protection against reflected XSS
- Content Security Policy (CSP)
  - → Domain-level control over resources to be included
  - → Most promising infrastructural technique against XSS
  - → Interesting reporting-only mode

Framing content securely






Overview

- Attacks:
  - → Click-jacking
  - → Same domain XSS
- Countermeasures:
  - → X-Frame-Options header
  - → 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

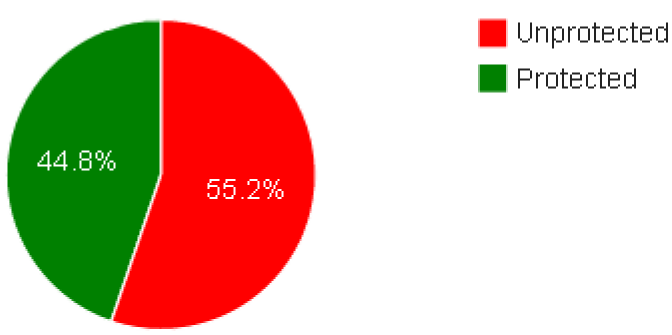
Source: "Busting Frame Busting: a Study of Clickjacking Vulnerabilities on Popular Sites" (W2SP 2010)

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

X-Frame-Options

- Browser compatibility:
  - → Firefox, Internet Explorer, Opera
  - → *Safari, Chrome*
- Usage statistics



	Domain	# of pages using X-Frame-Options	# of pages visited	Percentage of pages
1	facebook.com	500	500	100
2	t411.me	500	500	100
3	twitter.com	500	500	100
4	youtube.com	500	500	100
5	instagram.com	499	500	99
6	vimeo.com	498	500	99
7	linkedin.com	494	499	98
8	google.com	485	499	97
9	google.nl	480	500	96
10	google.be	428	500	85
11	google.fr	424	500	84
12	avg.com	391	499	78
13	live.com	381	463	82
14	kbc.be	310	500	62
15	paypal.com	301	399	75

Own experiment on top 100 of websites, visited from Belgium (Alexa)

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

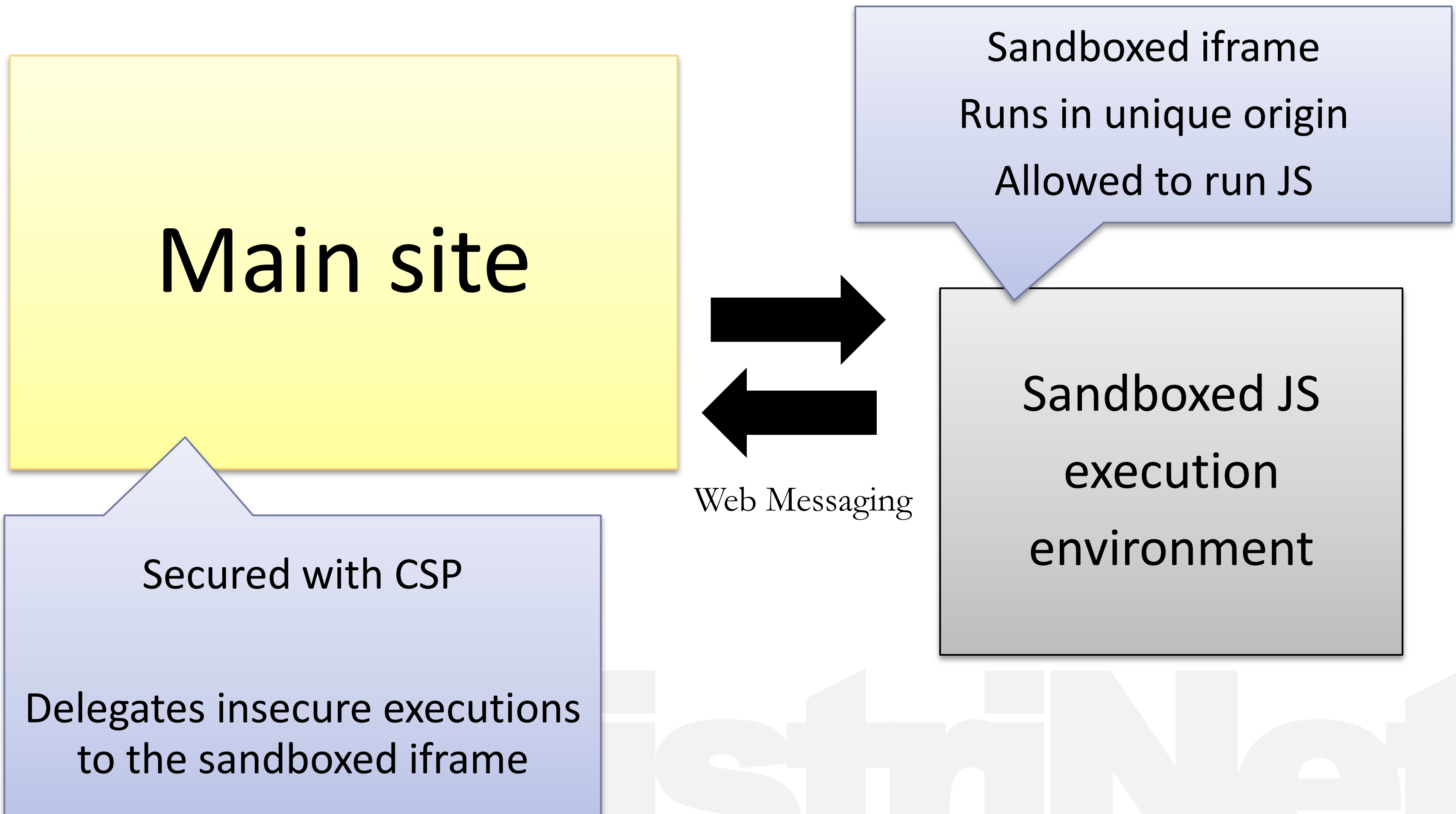
Sandbox relaxation directives

- 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

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 on Google docs, …

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>
```

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;
});
```

Sandboxed frame (sandbox.html)

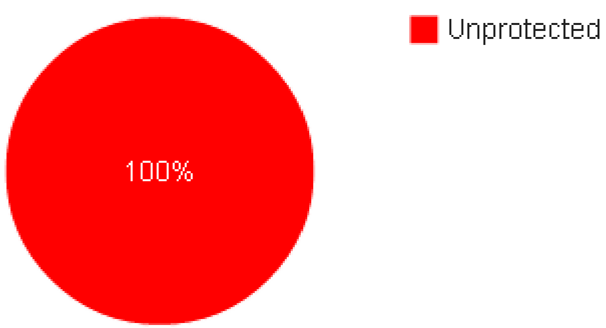
```
<html><head>
   <script>
         window.EventListener('message', function(event) {
          var command = event.data.command;
                var context = event.data.context;
                var result = callUnsafeFunction(command, context);
                event.source.postMessage({
        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

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

HTML5 sandbox

- Browser compatibility
  - → Internet Explorer, Chrome, Safari, Firefox
- Usage statistics



Own experiment on top 100 of websites, visited from Belgium (Alexa)

Recap: Framing content securely

- X-Frame-Options header
  - Robust defense against click-jacking
  - Any state-changing page should be protected
- HTML5 sandbox attribute for iframes
  - Coarse-grained sandboxing of resources and JavaScript
  - Interesting enabler for security architectures
    - More to come in the talk of Nick and Steven!

Enabling cross-domain interactions

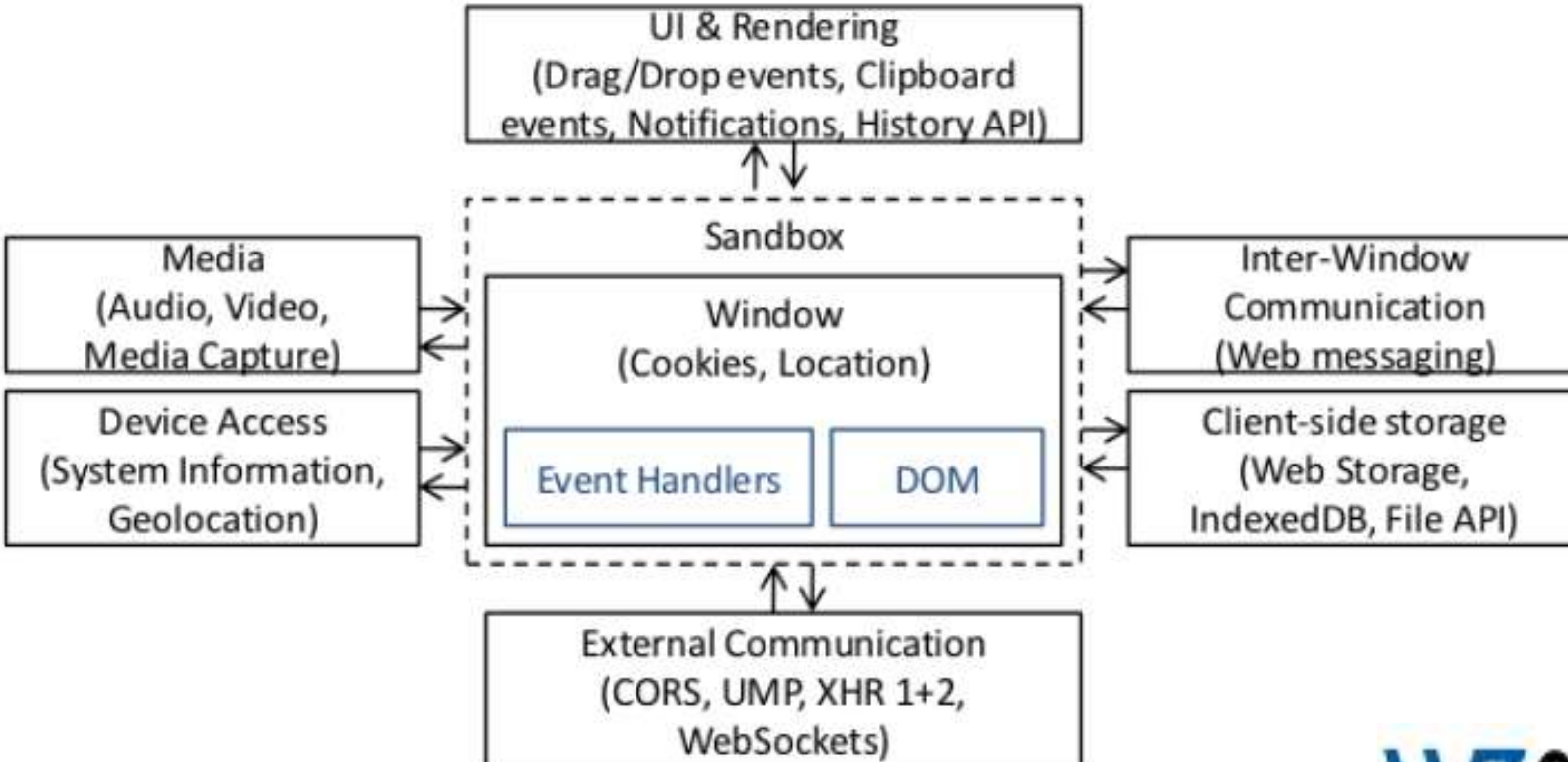





And there is a lot more …

- Problem:
  - Sometimes the Same-Origin Policy is too restrictive
- Enabling technologies:
  - Cross Origin Resource Sharing (CORS)
  - Crossdomain.xml
  - Web Messaging (aka postMessage)
  - …

HTML5: security analysis



Analysis of the specifications

- A Security Analysis of Next Generation Web Standards
  - → Commissioned by European Network and Information Security Agency (ENISA)
  - → Performed by iMinds-DistriNet, KU Leuven

enisa
European Network and Information Security Agency

- Full report available at ENISA
  - → http://www.enisa.europa.eu/activities/Resilience-and-CIIP/critical-applications/web-security/a-security-analysis-of-next-generation-web-standards

Analysis results

	Well-defined / Secure	Isolation Properties	Consistency	User Involvement
HTML5	8	3	2	2
Web Messaging		1	2	
XMLHttpRequest 1 + 2	1			
CORS	2	1		
UMP				
Web Storage	3	1	1	
Geolocation API	5	1	1	1
Media Capture API			3	
System Information API	3	1	1	2
Widgets - Digital Signatures				2
Widgets - Access Req Policy	3			1
Total	**25**	**8**	**10**	**8**

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…

Acknowledgements

- The work is partially funded by the European FP7 projects WebSand, STREWS and NESSoS.

SEVENTH FRAMEWORK PROGRAMME

WebSand

NESSoS

STREWS

- With the financial support from the Prevention of and Fight against Crime Programme of the European Union.

b·ccentre

BELGIAN CYBERCRIME CENTRE OF EXCELLENCE FOR TRAINING, RESEARCH & EDUCATION

References

- Ph. De Ryck, M. Decat, L. Desmet, F. Piessens, W. Joosen. Security of web mashups: a survey (NordSec 2010)
- G.Rydstedt, E. Bursztein, D. Boneh, and C. Jackson. Busting frame busting: a study of clickjacking vulnerabilities at popular sites (W2SP 2010)
- 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. 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)
- Mike West. Securing the Client-Side: Building safe web applications with HTML5 (Devoxx 2012)
- 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)