Recent Web Security Technology

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

SecAppDev Leuven 2015 (26/02/2015, Leuven)
About myself: 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 2015 Conference
- Program director at SecAppDev
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
  - Building blocks of the Web
  - 7 attacker models, broken down in 10 capabilities
  - 13 attacks and their countermeasures

- Provides an up-to-date overview of
  - 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
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.
Recent security technology on the web

Web Browser → HTTP Request → HTTP Response → Web Server

Policy enforcement in the browser

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

Iframe integration

```html
<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

HTTP Request

Cookie: PREF=ID=766awg-VZ

HTTP Response

HTTP Request

Cookie: PREF=ID=766awg-VZ

HTTP Response
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

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

Internet SSL Survey 2010, Qualys
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

Own experiment on top 2500 websites, visited from Belgium (Alexa)
Some background on this experiment

- Number of inspected domains: 2449
- Total number of inspected pages: 302855
- Average number of pages per domains: 123
- 18.25% of domains serve HTTPS pages
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)

*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 domains, grouped by 10,000

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)

Alexa Top 100,000 domains, grouped by McAfee’s site categories

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

HTTP Request

Redirect to HTTPS

HTTP Response

HTTPS Request

HTTPS Response

Web Browser

Web Server
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

HTTP Request
HTTP Response

HTTP Request
HTTP Response

HTTP Request
HTTPS Request
HTTPS Response

Redirect to HTTPS

Web Browser
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: state-of-practice

- **Browser compatibility**
  - Chrome 4+, Firefox 4+, Opera 12+, Safari 7+, IE12

- **Usage statistics**

  Own experiment on top 2500 websites, visited from Belgium (Alexa)
But can I trust the CAs?

- Comodo (March 2011)
  - 9 fraudulent 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=500; pin-sha1="4n972HfV354KP560yw4uqe/baXc="; pin-sha1="IvGeLsbqzPxdI0b0wuj2xVTdXgc="

- Freezes the certificate by pushing a fingerprint of (parts of) the certificate chain to the browser
  - Options: includeSubdomains, report-uri

- Currently an IETF Internet-Draft
  - Public Key Pinning Extension for HTTP

- Supported in Chrome 18+, Firefox 35+
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
#2 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

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

Regular http request

Http response containing script as part of executable content

Attacker

Victim

Vulnerable server

HTTP response
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

- Usage statistics

Own experiment on top 2500 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

<table>
<thead>
<tr>
<th>Domain</th>
<th># of pages using x_xss_protection</th>
<th># of pages visited</th>
<th>Percentage of pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 etsy.com</td>
<td>170</td>
<td>171</td>
<td>99.4152</td>
</tr>
<tr>
<td>2 google.com</td>
<td>151</td>
<td>154</td>
<td>98.0519</td>
</tr>
<tr>
<td>3 google.it</td>
<td>166</td>
<td>169</td>
<td>98.2249</td>
</tr>
<tr>
<td>4 search-results.com</td>
<td>144</td>
<td>170</td>
<td>84.7059</td>
</tr>
<tr>
<td>5 google.de</td>
<td>173</td>
<td>173</td>
<td>100</td>
</tr>
<tr>
<td>6 google.fr</td>
<td>164</td>
<td>164</td>
<td>100</td>
</tr>
<tr>
<td>7 google.es</td>
<td>156</td>
<td>158</td>
<td>98.7342</td>
</tr>
<tr>
<td>8 google.co.uk</td>
<td>150</td>
<td>151</td>
<td>99.3377</td>
</tr>
<tr>
<td>9 vroom.be</td>
<td>158</td>
<td>177</td>
<td>89.2855</td>
</tr>
<tr>
<td>10 google.co.in</td>
<td>168</td>
<td>168</td>
<td>100</td>
</tr>
</tbody>
</table>

Own experiment on top 2500 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
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 embedded 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

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

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

External JS

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

JavaScript code

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

Binding to page
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
    ```html
    <script nonce="Nc3n83cnSAd3wc3Sasdfn939hc3"> 
    alert("Allowed because nonce is valid."); 
    </script>
    ```
  - Add a hash of your inline script to the policy
    ```html
    <script> 
    alert('Hello, world.'); 
    </script> 
    ```
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

{  
  "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"
  }
}
CSP Reporting: one step further

- Apart from reporting violations via the report-uri directive

- CSP can also run in report only mode

  - 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!

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


Based on “HTML5Rocks: An introduction to Content Security Policy” (Mike West)
X-WebKit-CSP: default-src *;
script-src https://*.facebook.com http://*.facebook.com
chrome-extension://lifbcibllhkdhafpjfnlhfpgnpldfl '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

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

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

- **Browser compatibility:**
  - Firefox 4, Chrome 14+, Safari 5+, Opera 15+, Internet Explorer 10+
  - Older header names: X-WebKit-CSP, X-Content-Security-Policy

- **Usage statistics**

  Own experiment on top 2500 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
#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
X-Frame-Options (deprecated)

- Browser compatibility:
  - Firefox, Internet Explorer, Opera
  - Safari, Chrome

- Usage statistics

<table>
<thead>
<tr>
<th>Domain</th>
<th># of pages using X-Frame-Options</th>
<th># of pages visited</th>
<th>Percentage of pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>equibel.be</td>
<td>158</td>
<td>158</td>
<td>100</td>
</tr>
<tr>
<td>etsy.com</td>
<td>170</td>
<td>171</td>
<td>99.4152</td>
</tr>
<tr>
<td>soundcloud.com</td>
<td>166</td>
<td>173</td>
<td>95.9538</td>
</tr>
<tr>
<td>replacedirect.be</td>
<td>165</td>
<td>165</td>
<td>100</td>
</tr>
<tr>
<td>google.it</td>
<td>137</td>
<td>169</td>
<td>81.0651</td>
</tr>
<tr>
<td>napoleongames.be</td>
<td>142</td>
<td>145</td>
<td>97.931</td>
</tr>
<tr>
<td>bonprix-wa.be</td>
<td>176</td>
<td>177</td>
<td>99.435</td>
</tr>
<tr>
<td>dropbox.com</td>
<td>105</td>
<td>108</td>
<td>97.2222</td>
</tr>
<tr>
<td>csj.be</td>
<td>172</td>
<td>175</td>
<td>98.2857</td>
</tr>
<tr>
<td>facebook.com</td>
<td>60</td>
<td>63</td>
<td>95.2381</td>
</tr>
</tbody>
</table>

Own experiment on top 2500 websites, visited from Belgium (Alexa)
XFO has been integrated in CSP

- 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

- Browser compatibility
  - Internet Explorer, Chrome, Safari, Firefox, Opera

- Usage statistics

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

Secured with CSP
Delegates insecure executions to the sandboxed iframe

Sandboxed iframe
Runs in unique origin
Allowed to run JS

Sandboxed JS execution environment

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

```html
<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 id="#content"></div>
  </body>
</html>
```

“Securing the Client-Side: Building safe web applications with HTML5” (Mike West, Devoxx 2012)
<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>
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
  
  ```html
  <iframe sandbox seamless srcdoc="<p>Some paragraph</p>"/>
  ```

- 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…
Content Security Policy 1.0 - CR

Mitigate cross-site scripting attacks by whitelisting allowed sources of script, style, and other resources.

The standard HTTP header is Content-Security-Policy, which is used unless otherwise noted.
References

- 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)
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)
- 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)
- Can I use … ?, http://caniuse.com/