Content Security Policy

Ian Oxley, OWASP Newcastle, 29 September 2015

What is Content Security Policy?

Policy Directives, CSP Level 1, and CSP Level 2

Reporting

Security is hard

selfie with security guards by arileu: https://flic.kr/p/xaoQUS

XSS ranked in the top 3 vulnerabilities on the OWASP Top 10 since forever *

* 2007, 2010, and 2013

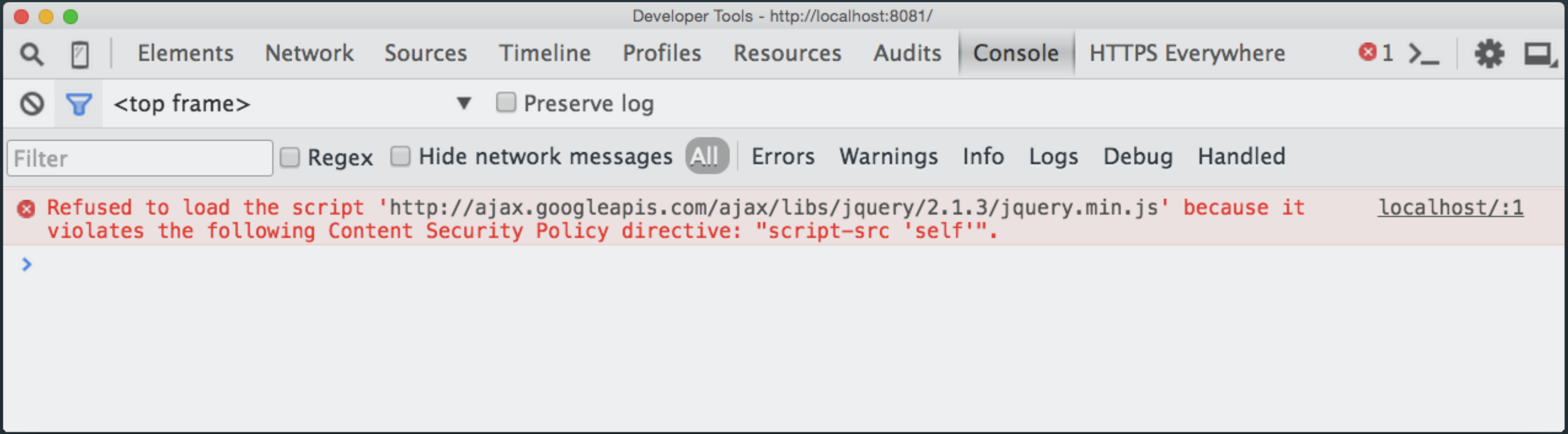
Content Security Policy:

Gives the browser a **whitelist of trusted sources** where content can be loaded or executed from

HTTP Header

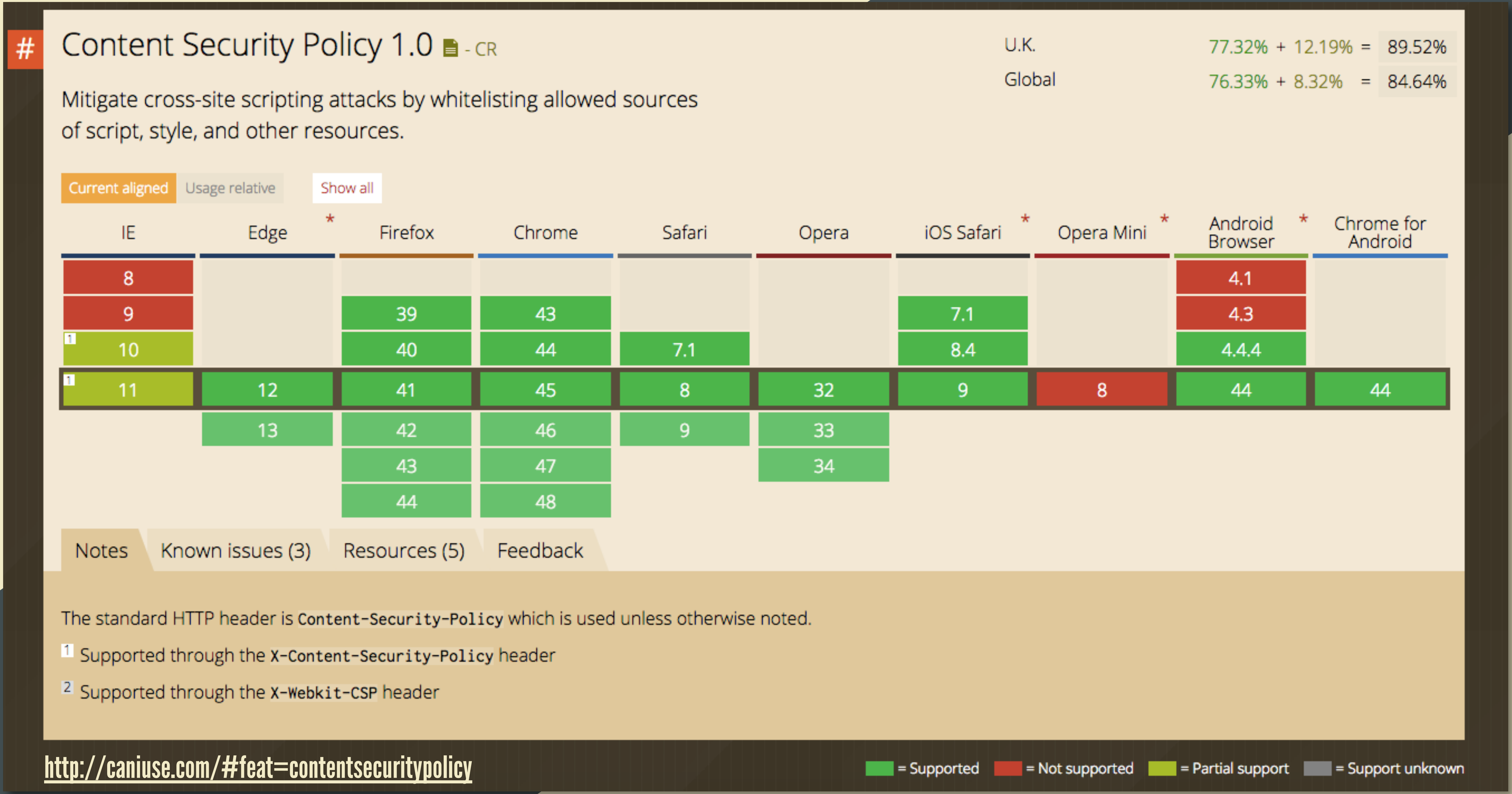
Content-Security-Policy: script-src ‘self’;

One or more directives



```
Refused to load the script 'http://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js' because it    localhost/:1
violates the following Content Security Policy directive: "script-src 'self'".
```

Level 1



Content Security Policy 1.0 - CR

U.K.	77.32%	+	12.19%	=	89.52%
Global	76.33%	+	8.32%	=	84.64%

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

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
🟥 8								🟥 4.1	
🟥 9		🟩 39	🟩 43			🟩 7.1		🟥 4.3	
▨ [1] 10		🟩 40	🟩 44	🟩 7.1		🟩 8.4		🟩 4.4.4	
▨ [1] 11	🟩 12	🟩 41	🟩 45	🟩 8	🟩 32	🟩 9	🟥 8	🟩 44	🟩 44
	🟩 13	🟩 42	🟩 46	🟩 9	🟩 33				
		🟩 43	🟩 47		🟩 34				
		🟩 44	🟩 48						

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

[1] Supported through the `X-Content-Security-Policy` header

[2] Supported through the `X-Webkit-CSP` header

http://caniuse.com/#feat=contentsecuritypolicy

🟩 = Supported 🟥 = Not supported ▨ = Partial support ⬜ = Support unknown

Level 1

default-src
style-src
connect-src
object-src
script-src
img-src
font-src
media-src
frame-src

‘self’

‘none’

URL

* wildcard support

data:

https:

https://cdn.example.com

://.example.com:*

```
<script>
  document.documentElement.className = ‘js’;
</script>
```

```
<style>
  .container {
    margin-top: 2rem;
    ...
  }
</style>
```

```
<section style=“margin-top: 1rem;”>
  ...
</section>
```

```
<a href=“javascript:link();”>...</a>
<img onclick=“loadPreview()”>
```

```
<script>
  document.documentElement.className = ‘js’;
</script>
```

```
<style>
  .container {
    margin-top: 2rem;
    ...
  }
</style>
```

```
<section style=“margin-top: 1rem;”>
  ...
</section>
```

```
<a href=“javascript:link();”>...</a>
<img onclick=“loadPreview()”>
```

‘unsafe-inline’

```
<script>
  var foo = new Function('foo', 'bar', 'return foo + bar');
</script>
```

```
<script>
  eval('console.log("foo")');
</script>
```

```
<script>
  setTimeout('console.log(foo);',
    5000);
  setInterval('console.log(foo);',
    5000);
</script>
```

```
<script>
  var foo = new Function('foo', 'bar', 'return foo + bar');
</script>
```

```
<script>
  eval('console.log("foo")');
</script>
```

```
<script>
  setTimeout('console.log(foo);',
    5000);
  setInterval('console.log(foo);',
    5000);
</script>
```

'unsafe-eval'

```
Content-Security-Policy:
        default-src ‘self’;
```

```
Content-Security-Policy:
        default-src ‘self’ https:;
```

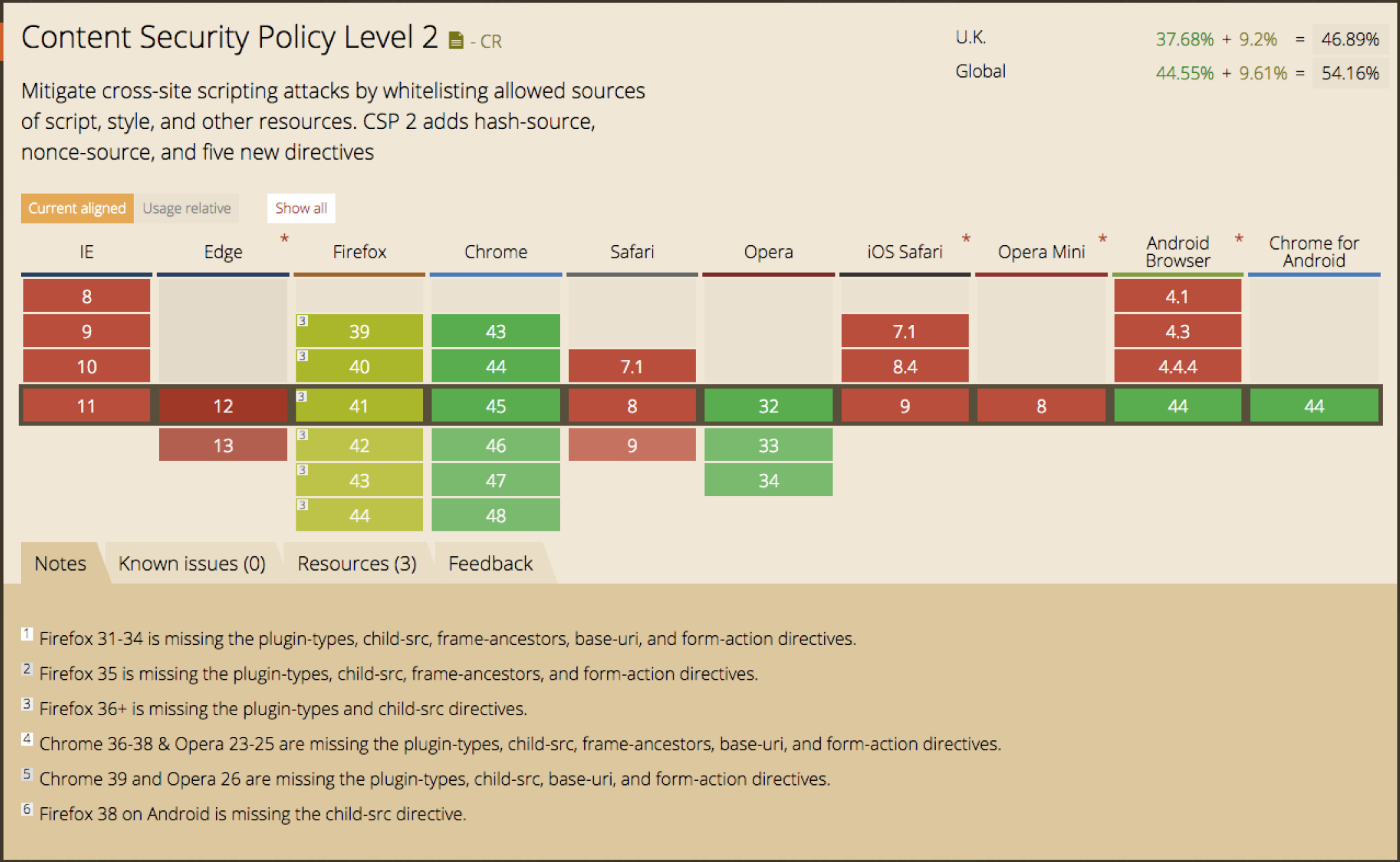
```
Content-Security-Policy:
        default-src ‘self’ https:;
        script-src https://cdn.example.com
```

```
Content-Security-Policy:
        default-src ‘self’ https:;
        script-src ‘self’ https: https://cdn.example.com
```

```
Content-Security-Policy:
        default-src ‘self’ https:;
        script-src ‘self’ https: https://cdn.example.com
                              https://ajax.googleapis.com;
        style-src ‘self’ https: https://cdn.example.com;
```

```
Content-Security-Policy:
        default-src ‘self’ https:;
        script-src ‘self’ https: https://cdn.example.com
                             https://ajax.googleapis.com;
        style-src ‘self’ https: https://cdn.example.com;
        object-src ‘none’;
```

Level 2



Content Security Policy Level 2 - CR

U.K.	37.68%	+ 9.2%	=	46.89%
Global	44.55%	+ 9.61%	=	54.16%

Mitigate cross-site scripting attacks by whitelisting allowed sources of script, style, and other resources. CSP 2 adds hash-source, nonce-source, and five new directives

Current aligned Usage relative Show all

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
🟥 8								🟥 4.1	
🟥 9		🟨 39 [3]	🟩 43			🟥 7.1		🟥 4.3	
🟥 10		🟨 40 [3]	🟩 44	🟥 7.1		🟥 8.4		🟥 4.4.4	
🟥 11	🟥 12	🟨 41 [3]	🟩 45	🟥 8	🟩 32	🟥 9	🟥 8	🟩 44	🟩 44
	🟥 13	🟨 42 [3]	🟩 46	🟥 9	🟩 33				
		🟨 43 [3]	🟩 47		🟩 34				
		🟨 44 [3]	🟩 48						

Notes | Known issues (0) | Resources (3) | Feedback

[1] Firefox 31-34 is missing the plugin-types, child-src, frame-ancestors, base-uri, and form-action directives.

[2] Firefox 35 is missing the plugin-types, child-src, frame-ancestors, and form-action directives.

[3] Firefox 36+ is missing the plugin-types and child-src directives.

[4] Chrome 36-38 & Opera 23-25 are missing the plugin-types, child-src, frame-ancestors, base-uri, and form-action directives.

[5] Chrome 39 and Opera 26 are missing the plugin-types, child-src, base-uri, and form-action directives.

[6] Firefox 38 on Android is missing the child-src directive.

http://caniuse.com/#feat=contentsecuritypolicy2

🟩 = Supported 🟥 = Not supported 🟨 = Partial support ⬜ = Support unknown

Level 2

default-src
style-src
connect-src
object-src
script-src
img-src
font-src
media-src
frame-src

Level 2

default-src

frame-ancestors

style-src

plugin-types

connect-src

object-src

base-uri

script-src

child-src

img-src

font-src

form-actions

media-src

~~frame-src~~

<meta>

```
<meta http-equiv="Content-Security-Policy"
      content="default-src 'self' https:;">
```

Using a nonce

```
Content-Security-Policy:
        default-src ‘self’;
        script-src ‘self’ https://example.com
                    ‘nonce-X87di93dkeff’;
```

```
<script>
  console.log(
    “No nonce attribute - won’t execute”);
</script>
```

```
<script nonce=“Gdidj89sk28j92pp”>
  console.log(
    “Nonce mismatch - won’t execute”);
</script>
```

```
<script nonce="X87di93dkeff">
   console.log(
      "Nonce matches - script executes");
</script>
```

```
// Valid nonce - script executes
<script nonce="X87di93dkeff"
      src="//url.com/not-on-whitelist">
</script>
```

Using a hash

```
Content-Security-Policy:
        default-src ‘self’;
        script-src ‘self’
        ‘sha256-2eXTeAxXc4NfEdtTitmpuNQV
        41/dtCeCYiAwxZCvkGo=’;
```

```
// Script executes - computed hash
// matches the one in the header
// matches the one in the header
<script>alert('Hello, OWASP.');</script>
```

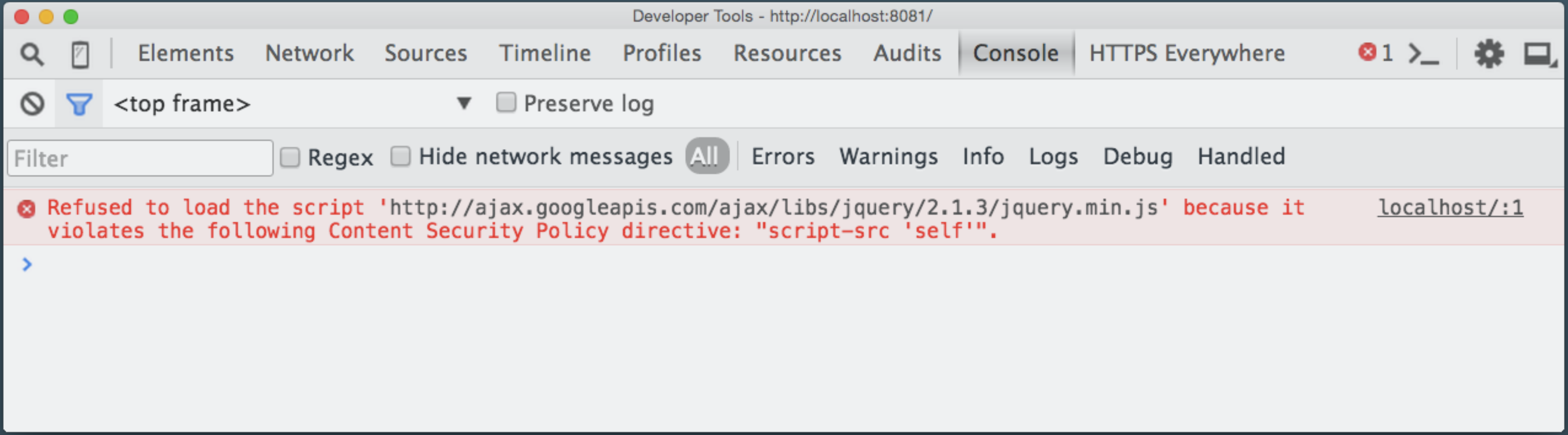
```
// Neither of these execute - whitespace
// and newlines cause a different hash
// to be computed
<script>  alert('Hello, OWASP.');</script>

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

" If you really, absolutely must have inline script and style, you can enable it by adding 'unsafe-inline' as an allowed source in a script-src or style-src directive. You can also use a nonce or a hash (see below). But please don't. Banning inline script is the biggest security win CSP provides, and banning inline style likewise hardens your application.

Mike West, An Introduction to Content Security Policy
http://www.html5rocks.com/en/tutorials/security/content-security-policy/

Reporting

Browser consoles don’t work so well in production



```
Refused to load the script 'http://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js' because it    localhost/:1
violates the following Content Security Policy directive: "script-src 'self'".
```

report-uri directive takes a **URL** as its value. **JSON** sent via **HTTP POST** for each policy violation.

```
Content-Security-Policy:
                    default-src ‘self’;
                    script-src ‘self’ http://cdn.example.com;
                    style-src ‘self’ http://cdn.example.com;
                    report-uri /csp-report;
```

```
Content-Security-Policy-Report-Only:
    default-src ‘self’ https:; https://cdn.example.com
    report-uri /csp-report;
```

```
Content-Security-Policy:
            default-src ‘self’;
            script-src ‘self’ http://cdn.example.com;
            style-src ‘self’ http://cdn.example.com;
            report-uri /csp-report;

Content-Security-Policy-Report-Only:
      default-src ‘self’ https:; https://cdn.example.com
      report-uri /csp-report;
```

CSP whitelists trusted origins using policy directives

Although you can enable them, inline styles and scripts are off by default for a reason

report-uri can monitor CSP in production any questions?

@ianoxley

thanks

@ianoxley