



**The OWASP Foundation**  
<http://www.owasp.org>

# Content Security Policy (CSP)

OWASP Stammtisch Stuttgart



# Vorwort

Mit freundlicher Genehmigung von Christine Koppelt  
(Danke an der Stelle 😊)

Dieser Vortrag basiert auf Ihrem vom 19.02.13 beim  
OWASP Stammtisch München

Aber: mangels Zeit zur Vorbereitung keine Demo 😞

<http://securityreactions.tumblr.com/post/45185865476/live-pwning-during-a-presentation>



# Übersicht

Warum?

CSwas?

Beispiele (ohne Demo)

Probleme

Möglicher Nutzungsansatz der CSP





# Warum

Cross-Site-Scripting (XSS) ist auf Platz

2 (2010)

3 (2013 RC)

Aufwändige (und fehleranfällige) Prävention  
trotz Hilfsmitteln (Frameworks, Filter, ...)



# CSP

- Im Browser implementierte Richtlinie, um möglichst granular Quellen für Inhalte zu deklarieren.
- Design achtet auf Abwärtskompatibilität
- Dadurch:
  - Verhinderung der Ausführung von Inline-JavaScript
  - White-Listing externer Quellen
    - CSS, Images, Fonts, (i)Frames, XSLT, Video, Audio



# CSP - Technisch

## HTTP-Header-Key:

- Offiziell: Content-Security-Policy
- Firefox: X-Content-Security-Policy
- Chrome: X-WebKit-CSP

## HTTP-Header-Value:

- Besteht wiederum aus Key-Value-Paaren
- Festlegung der Einschränkung der einzelnen Ressourcentypen



# CSP – Stand der Dinge

Konzeptionell ursprünglich von Mozilla  
Foundation und Google entwickelt

W3C Draft (1.0 mit Stand 15.11.2012, 1.1  
wird aktuell rege bearbeitet)

# CSP – Stand der Dinge

**Content Security Policy- Working Draft**

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

Resources: [HTML5Rocks article](#)

Global user stats\*:

Support:	57%
Partial support:	2.58%
Total:	59.58%

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android
21 versions back			4.0									
20 versions back			5.0									
19 versions back		2.0	6.0									
18 versions back		3.0	7.0									
17 versions back		3.5	8.0									
16 versions back		3.6	9.0									
15 versions back		4.0 moz	10.0									
14 versions back		5.0 moz	11.0									
13 versions back		6.0 moz	12.0									
12 versions back		7.0 moz	13.0									
11 versions back		8.0 moz	14.0 webkit									
10 versions back		9.0 moz	15.0 webkit		9.0							
9 versions back		10.0 moz	16.0 webkit		9.5-9.6							
8 versions back		11.0 moz	17.0 webkit		10.0-10.1							
7 versions back		12.0 moz	18.0 webkit		10.5							
6 versions back		13.0 moz	19.0 webkit		10.6			2.1				
5 versions back	5.5	14.0 moz	20.0 webkit	3.1	11.0			2.2		10.0		
4 versions back	6.0	15.0 moz	21.0 webkit	3.2	11.1	3.2		2.3		11.0		
3 versions back	7.0	16.0 moz	22.0 webkit	4.0	11.5	4.0-4.1		3.0		11.1		
2 versions back	8.0	17.0 moz	23.0 webkit	5.0	11.6	4.2-4.3		4.0		11.5		
Previous version	9.0	18.0 moz	24.0 webkit	5.1 webkit	12.0	5.0-5.1		4.1		12.0		
Current	10.0 ms	19.0 moz	25.0	6.0 webkit	12.1	6.0 webkit	5.0-7.0	4.2	7.0	12.1	25.0 webkit	19.0 moz
Near future		20.0 moz	26.0		12.5				10.0			
Farther future		21.0 moz	27.0									

Source: caniuse.com – March 2013



# Beispiele

```
Content-Security-Policy: default-src  
  'self'
```

Ressourcen dürfen nur von derselben Domain geladen\* werden, inline Deklarationen sind ausgeschlossen (\*=auch Subdomains werden ausgeschlossen)

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


Externe Ressourcen dürfen nur über https geladen werden, inline Deklarationen sind zulässig



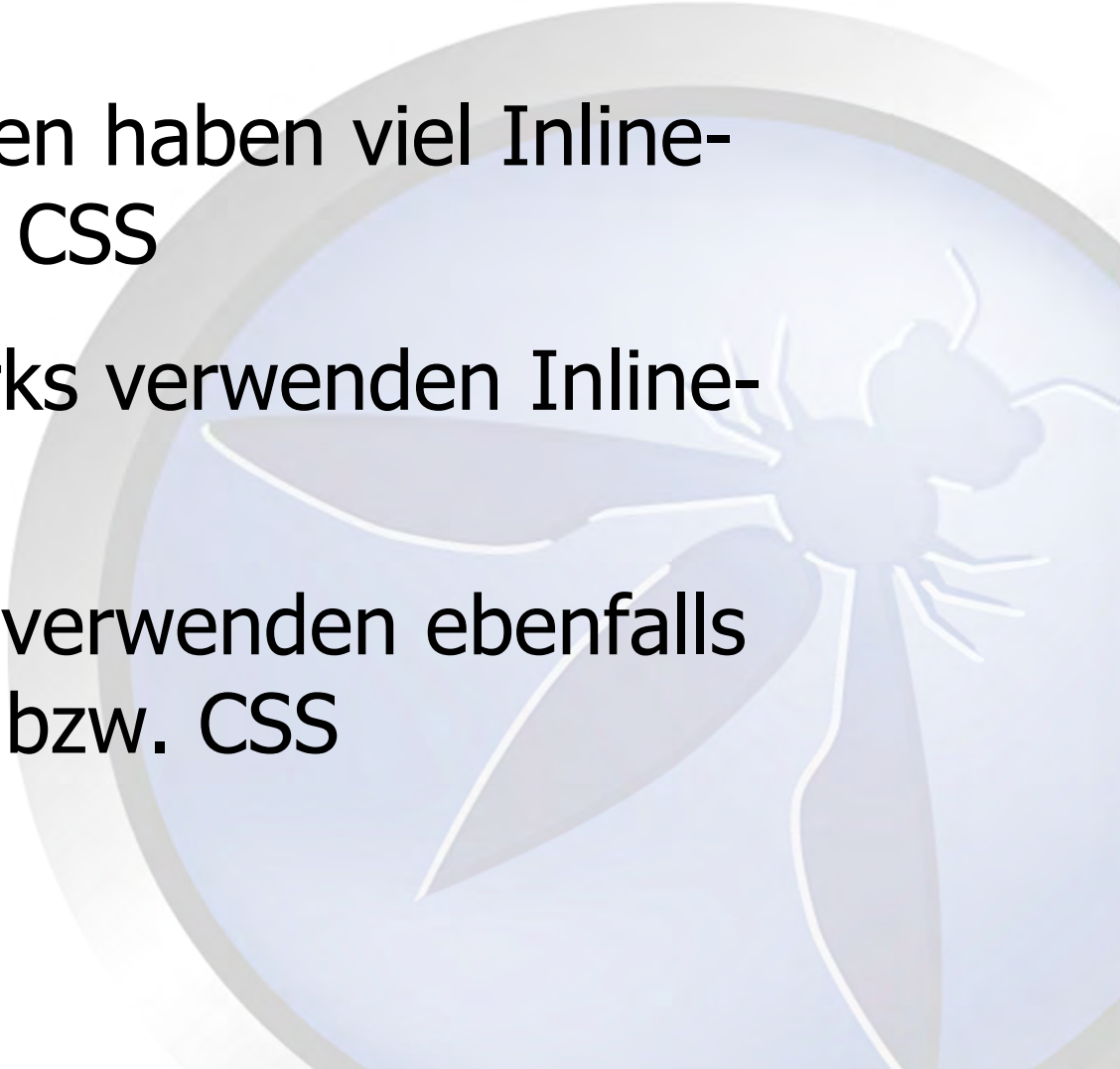
# Beispiele (2)

```
Content-Security-Policy: default-src  
  'self'; img-src *; script-src  
  trusted.example.com
```

Bilder dürfen von überall geladen werden, Skripte von trusted.example.com, alle anderen Ressourcen nur von der derselben Domain



# Probleme

- Alte Anwendungen haben viel Inline-  
JavaScripts bzw. CSS
  - Selbst Frameworks verwenden Inline-  
Scripts
  - Browser-Plugins verwenden ebenfalls  
inline JavaScript bzw. CSS
- 



# Nutzungsansatz

- Verständnis der Direktiven und deren Auswirkungen auf die Anwendung im Browser
- Zuerst: Nutzung der Violation-Report-Funktion
  - Content-Security-Policy-Report-Only
  - Browser sendet JSON-Objekt mit Violation-Report per HTTP-POST an eine festzulegende URI
  - Script zum Verarbeiten der Reports muss (sicher) entwickelt werden
  - Erweiterung der Policy-Direktiven um „report-uri <URL Report-Script>“

# Beispiel

## Violation-Report:

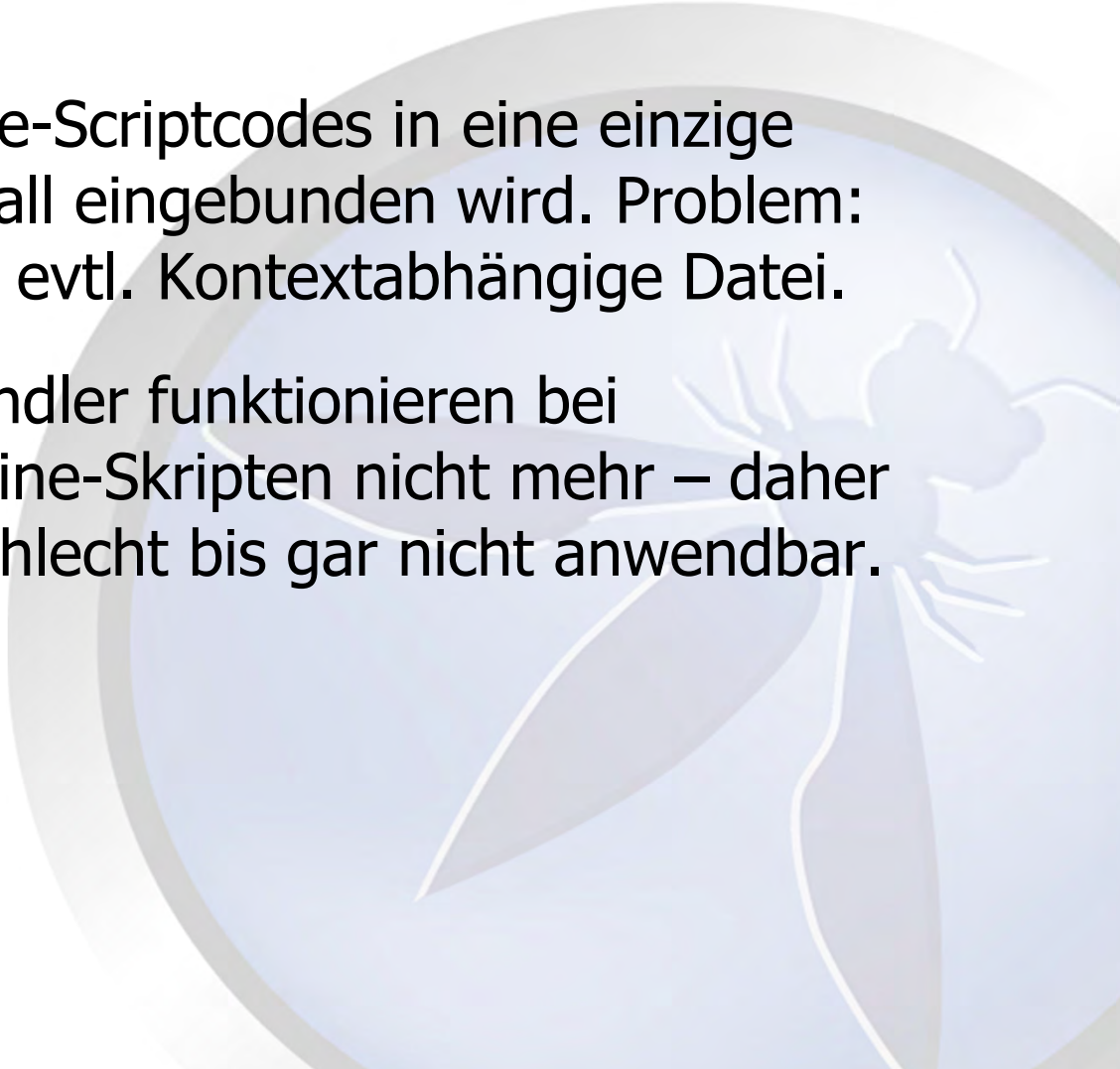
```
default-src 'self'; report-uri  
  http://example.org/csp-report.cgi
```

Bild von <http://evil.example.com/image.png> verstößt gegen die Policy

```
{ "csp-report": {  
  "document-uri": "http://example.org/page.html",  
  "referrer": "http://evil.example.com/haxor.html",  
  "blocked-uri": "http://evil.example.com/image.png",  
  "violated-directive": "default-src 'self'",  
  "original-policy": "default-src 'self';  
  report-uri http://example.org/csp-report.cgi" }  
}
```



# Nutzungsansatz (2)

- Verschieben des Inline-Scriptcodes in eine einzige Script-Datei, die überall eingebunden wird. Problem: Überfrachtung, daher evtl. Kontextabhängige Datei.
  - Problem: on-Eventhandler funktionieren bei Deaktivierung von Inline-Skripten nicht mehr – daher in vielen Szenarien schlecht bis gar nicht anwendbar.
- 



</Lightning-Talk>

Danke

<Discussion/>