URL:  
  
Category:   
Content Security Policy  
KWs:   
Content Security Policy, content security policy header, content security policy generator, content security policy beispiel, content security policy firefox  
Internal links:  
Yes  
Title:  
Content-Security-Policy: Wie Websites sicherer werden  
Description:  
Durch Content-Security-Policy nutzen Sie Firefox und Co. mit weniger Risiko. Was steckt hinter dem Sicherheitsstandard für Webbrowser?  
Number of words: Target / Actual  
Min200- Max1000 / 2059  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Content-Security-Policy: Mehr Sicherheit bei Webinhalten

Aktive Inhalte auf Websites – z. B. JavaScript, CSS, ActiveX – stellen ein Sicherheitsrisiko für Internetnutzer und Betreiber von Websites dar. Denn sie können durch sogenanntes Cross-Site-Scripting [Was ist XSS bzw. Cross-Site-Scripting?] (https://hosting.1und1.de/digitalguide/websites/web-entwicklung/was-ist-xss-bzw-cross-site-scripting/) manipuliert werden. Damit Internetseiten trotzdem ohne Bedenken in vollem Umfang genutzt werden können, hat man eine Content-Security-Policy (CSP) eingeführt. Der **Sicherheitsstandard** soll gegen schädliche Angriffe schützen und wird inzwischen von den meisten Webbrowsern unterstützt. Das Sicherheitskonzept schützt sowohl Websites als auch Internetnutzer. Aber was steckt dahinter und wie funktioniert CSP?

## Entwicklung der Content-Security-Policy

Die Content-Security-Policy hat ihre Ursprünge im Jahr 2004 – damals noch unter dem Namen „Content-Restriction“. Grund für die Bemühungen waren zunehmende Sicherheitslücken bei Internetskripten. Besonders das **Cross-Site-Scripting (XSS)**, eine kriminelle Methode, um Schadcode in eine Website einzuschleusen, stellt ein großes Risiko für Anwender dar. Dabei rufen die Nutzer zwar eine vertrauenswürdige Website auf, dort läuft allerdings ein Skript, das schädliche Daten aus einer fremden Quelle lädt. Die Cyberkriminellen nutzen dazu z. B. Sicherheitslücken in Kommentarfunktionen der Website. So können sie Zugang zu persönlichen Computern erlangen, ohne dass sich die Internetnutzer dessen überhaupt bewusst sind. Die Webmaster bemerken meist ebenso wenig, dass der fremde Code eingeschleust wurde.

Um dieses Problem zu beheben, hat die **Mozilla Foundation** die Entwicklung der CSP vorangetrieben. Der Vorteil hinter dem Sicherheitsstandard ist, dass im Browser Regeln aufgestellt werden können, welche Skripte die Software laden darf und welche nicht. Dazu setzt die Content-Security-Policy auf HTTP-Header.

[Fakt: Die Mozilla Foundation ist die Organisation hinter der Entwicklung des Firefox-Browsers. Die Non-Profit-Organisation kümmert sich um die Ausrichtung der Projekte von Mozilla und um Innovationen rund um das Internet. Gegründet wurde die Foundation von ehemaligen Mitarbeitern von Netscape, die einen der ersten Webbrowser entwickelt hatten.]

## Wie funktioniert die Content-Security-Policy?

Bei der Kommunikation im Internet tauschen Client und Server Daten über das Hypertext Transfer Protocol (HTTP) aus. Ein wichtiger Bestandteil der Requests und Responses sind **HTTP-Header-Felder**: In diesen werden Parameter und Argumente übermittelt, die für den Austausch der beiden Gesprächsteilnehmer (Server und Client) wichtig sind. Sie werden grundsätzlich in Anfrage- und Antwortfelder aufgeteilt und können z. B. Informationen über Zeichensatz, Sprache und Cookies beinhalten. CSP wird als Antwort-Headerfeld realisiert. Das bedeutet, dass der Server die Informationen liefert und der Browser diese verarbeitet.

Der Content-Security-Policy-Header wird vom Webmaster erstellt und auf jeder Unterseite der Website eingefügt, auf der der Sicherheitsstandard gelten soll. Als Betreiber der Website haben Sie so die Möglichkeit, **unterschiedliche Sicherheitsvorkehrungen für jede einzelne Seite** festzulegen. Unkomplizierter realisieren Sie das Sicherheitskonzept, indem Sie .htaccess-Dateien [Was ist eine .htaccess? Erklärung + 10 besten Tricks & Beispiele] (https://hosting.1und1.de/digitalguide/hosting/hosting-technik/die-zehn-besten-htaccess-tricks/) erstellen, die in den gleichen (oder hierarchisch höheren) Ordnern wie die entsprechenden Webpages liegen, oder indem Sie CSP direkt in die Server-Konfiguration verankern.

[Hinweis: Wenn Sie als Internetnutzer überprüfen möchten, ob Ihr Browser eine Content-Security-Policy unterstützt, können Sie den CSP-Browser-Test aufrufen [Content Security Policy Browser Test] (https://content-security-policy.com/browser-test/). Dieser versucht Skripts und Bilder aus fremden (unschädlichen) Quellen zu laden und zeigt Ihnen, ob dies erfolgreich war. Ihr Browser ist dann sicher, wenn er die fremden Quellen nicht lädt, also die CSP versteht und anwendet.]

Im Sinne einer Content-Security-Policy und um Cross-Site-Scripting zu verhindern, sollten Webmaster sämtliche Skripte in gesonderte Dateien auslagern, statt sie direkt im Quellcode der Homepage einzubetten. Die CSP funktioniert dann nach dem Prinzip einer **Whitelist**: Im Header werden die Quellen genannt, aus denen Skripte und Daten geladen werden dürfen. Sollte ein fremdes Skript unerkannt in den HTML-Code der Seite eingeschleust worden sein und nun versuchen, externe Daten zu laden, muss der Browser des Nutzers dieses verbieten. Standardmäßig blockiert eine CSP alle Skripte, die sich direkt im Code befinden (Inline-Skripte). So werden sowohl die Website als auch der Internetnutzer – und vor allem dessen sensible Daten – geschützt.

Manipulationen durch Cross-Site-Scripting sind für Cyberkriminelle relativ leicht durchzuführen. Fast jede Seite im World Wide Web besitzt ein Eingabefeld: z. B. Kommentarfunktion, Suchleiste oder Log-in-Feld. Statt einfachem Text können dort auch Skripte eingefügt werden. Wenn der Server nicht entsprechend abgesichert ist, implementieren Kriminelle auf diesem Weg Phishing-Interfaces [Identitätsdiebstahl im Internet] (https://hosting.1und1.de/digitalguide/websites/webseiten-erstellen/identitaetsdiebstahl-im-netz/), bringen die gesamte Website zum Erliegen oder erlangen durch Schadsoftware die Kontrolle über den Webbrowser des Nutzers. CSP (oder richtiger: das entsprechende Headerfeld) teilt dem Webbrowser mit, **aus welchen Quellen er Daten laden darf**. Ist die Policy im Code der Website implementiert, wird der Versuch, über XSS eingeschleusten Code abzurufen, mit einer Fehlermeldung beantwortet.

Über die Content-Security-Policy können Webmaster aber auch viele weitere Einstellungen vornehmen, beispielweise durch diese Direktiven:

* **base-uri**: Beschränkt die URLs, die im *<base>*-Element der Webpage auftauchen dürfen.
* **child-src**: Legt fest, aus welchen Quellen Daten in Frames auftauchen dürfen, z. B. bei eingebetteten Videos von Drittanbietern.
* **connect-src**: Beschränkt die Quellen, mit denen sich die Seite verbinden kann, z. B. über Links.
* **font-src**: Bestimmt die Quellen, aus denen Schriftarten geladen werden dürfen.
* **form-action**: Stellt eine Liste von gültigen Endpunkten in Formularen bereit.
* **frame-ancestors**: Legt fest, welche Domains die Seite in Frames und iFrames einbauen dürfen.
* **img-src**: Beschränkt die Quellen, aus denen Bilder geladen werden dürfen.
* **media-src**: Legt fest, aus welchen Quellen Audio- und Video-Formate geladen werden dürfen.
* **object-src**: Definiert die Kontrolle über Flash und andere Plug-ins.
* **plugin-types**: Limitiert die Arten von Plug-ins.
* **report-uri**: Spezifiziert eine URL, an die Berichte geschickt werden, wenn gegen die Sicherheitsmaßnahmen verstoßen wurde.
* **script-src**: Bestimmt, welche Quellen für JavaScript erlaubt sind.
* **style-src**: Funktioniert wie script-src, wird allerdings bei Stylesheets angewendet.
* **upgrade-insecure-requests**: Legt fest, dass unsichere Seiten mit HTTP wie HTTPS-Seiten behandelt werden.
* **sandbox**: Verschiebt die betreffende Seite in eine Sandbox, in der u. a. Formulare, Pop-ups und Skripte verboten sind.

Diese Direktiven gelten nur, wenn sie ausdrücklich gesetzt sind. Ansonsten stehen sie standardmäßig offen und stellen somit eine Sicherheitslücke dar. Mit der ***default-src*** lässt sich dies allerdings ändern: Hier können Sie den standardmäßigen Zustand aller Direktiven festlegen, die mit *-src* enden. Statt diese offen zu lassen, stellen Sie z. B. ein, dass nur Daten aus Ihrer eigenen Website geladen werden dürfen, es sei denn, Sie haben es im HTTP-Header einer einzelnen Webpage anders festgelegt. In gesonderten Direktiven fügen Sie dann noch weitere Quellen hinzu.

In das Header-Feld können Sie beliebig viele Direktiven eintragen. Möchten Sie mehrere Direktiven aufnehmen, trennen Sie diese durch Semikolons. Außerdem müssen Sie als Webmaster alle Quellen innerhalb einer Direktive angeben. Mehrmalige Nennungen von gleichen Direktiven mit zusätzlichen Quellen wie im folgenden Beispiel sind **nicht zulässig**:

*script-src beispiel1.local; script-src beispiel2.local*

In diesem Fall ist nur die erste Quelle relevant, die zweite würde vom Client ignoriert. Stattdessen müssen Sie **beide Quellen in einer Direktive notieren**:

*script-src beispiel1.local beispiel2.local*

Sollten Sie für eine Page oder die komplette Website bestimmte Inhaltstypen gar nicht brauchen, können Sie in die entsprechenden Direktiven den Wert ***'none'*** in den Header eintragen – so legen Sie fest, dass gar keine Quellen geladen werden dürfen. Sie können auch den Wert ***'self'*** verwenden – damit bestimmen Sie, dass der Browser nur Inhalte aus der gleichen Quelle nachladen darf. Beide Werte müssen Sie immer in einfachen Anführungszeichen notieren, ansonsten werden *none* und *self* als Domains interpretiert.

Um eine Content-Security-Policy zu definieren, gibt es unterschiedliche Header-Möglichkeiten:

* **Content-Security-Policy**
* **X-Webkit-CSP**
* **X-Content-Security-Policy**

Nicht alle Browser unterstützen auch jede Benennung. Das W3C (das Gremium, um Standards im Web festzulegen) schlägt allerdings **Content-Security-Policy** vor [Zum CSP-Level-3-Dokument der W3C] (https://www.w3.org/TR/CSP3/). Deshalb haben sich inzwischen **alle modernen Browser** diesem Sicherheitsstandard angepasst (die anderen beiden Versionen gelten als veraltet). Um sicherzugehen, dass Sie einen möglichst großen Teil von Internetnutzern (auch diejenigen mit veralteten Browser-Versionen) mit Ihrer CSP erreichen, ist es durchaus ratsam, alle Headerfelder einzubauen. Kann ein entsprechender Webbrowser nichts mit dem Content-Security-Policy-Header anfangen, wird er diesen einfach ignorieren und die Website ohne Probleme darstellen – der zusätzliche Schutz ist für die betreffenden Nutzer dann allerdings nicht gegeben.

Für gewöhnlich werden Sie HTTP-Header seitenübergreifend für Ihre gesamte Domain setzen. Für Unterverzeichnisse können Sie die .htaccess-Datei verwenden. Mithilfe des CSP-Sicherheitsstandards legen Sie dann **Sonderregeln für einzelne Unterseiten** fest. Wenn Sie beispielsweise auf einer Seite einen Social-Media-Button implementiert haben, auf der nächsten aber nicht, ist es sinnvoll, auch nur auf der ersten Seite den Zugang zur fremden Quelle zu erlauben.

Quellen lassen sich als Adressen, in ihrer Art oder auch als Wildcards eingeben. Folgende Eingaben sind also zulässig:

* **script-src https://example.com:443** – Skripte sind nur von dieser Domain per HTTPS zulässig.
* **script-src 'none'** – Skripte dürfen nicht geladen werden.
* **script-src 'self'** – Skripte dürfen aus demselben Ursprung wie die aktuelle Seite geladen werden, nicht aber von Subdomains.
* **script-src https:** – Skripte dürfen von jeglicher Domain geladen werden, solange sie mit HTTPS beginnt.
* **script-src example.com** – Skripte dürfen von dieser Domain geladen werden.
* **script-src \*.example.com** – Skripte von dieser Domain und von allen Subdomains sind zulässig.
* **img-src data:** – Bilder dürfen über Data-URLs geladen werden.

Eine Content-Security-Policy legt grundlegend fest, dass Skripte nur aus Dateien geladen werden dürfen, nicht direkt aus dem Code der Website. Möchten Sie dies umgehen, können Sie den Befehl ***script-src 'unsafe-inline'***verwenden. Es sollte Ihnen aber bewusst sein, dass Sie damit wiederum eine Sicherheitslücke schaffen. Der Sicherheitsstandard verbietet überdies die Funktion *eval ()*. Eigentlich kann man mit diesem Befehl Text in JavaScript-Code umwandeln – doch auch das stellt ein Sicherheitsrisiko dar. Wer diese Funktion dennoch braucht, kann sie mit ***script-src 'unsafe-eval'***wieder aktivieren.

[Tipp: Über einen Umweg können Sie *unsafe-inline* doch sichern. Durch Hash-Werte oder Nonce-Source schließen Sie diese Sicherheitslücke weitestgehend [Zur Entwicklerseite von Mozilla] (https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/script-src#Unsafe\_inline\_script).]

Wenn Skripte nicht mehr im Code direkt auftauchen dürfen, müssen Sie **für jedes Skript eine eigene Datei anlegen**. Die Funktion des Skripts wird in eine .js-Datei ausgelagert. Im Code der Website wird nur noch auf diese verwiesen:

*<script src='beispiel.js'></script>*

Was das Skript schließlich macht, steht in der *beispiel.js*. Auch Style-Elemente müssen Sie in gesonderte Stylesheets auslagern. Wenn Sie als Webmaster also eine Internet-Security-Policy einführen möchten, reicht es nicht, den Header einzufügen. Sie müssen auch den **Quellcode Ihrer Website überprüfen und anpassen**.

[Hinweis: Sie möchten wissen, wie sicher Ihre eigene Website ist? Mozilla stellt einen einfach zu nutzenden Test bereit: Observatory by Mozilla [Zum Scan von Mozilla] (https://observatory.mozilla.org) vergibt nach Abschluss eines Scans eine Note und teilt Ihnen mit, an welchen Punkten Sie Ihre Website sicherer machen sollten.]

## Content-Security-Policy: Ein Beispiel

An einem Beispiel zeigen wir nun, wie Sie einen Content-Security-Policy-Header realisieren können und erklären, was mit dem Header erreicht wird.

*Content-Security-Policy: "default-src 'none'; script-src 'self' \*.example.com; style-src 'self'; img-src 'self' data:; font-src 'self' https://fonts.google.com/; report-uri https://example.org/report.html"*

*X-Content-Security-Policy: "default-src 'none'; script-src 'self' \*.example.com; style-src 'self'; img-src 'self' data:; font-src 'self' https://fonts.google.com/; report-uri https://example.org/report.html"*

*X-WebKit-CSP: "default-src 'none'; script-src 'self' \*.example.com; style-src 'self'; img-src 'self' data:; font-src 'self' https://fonts.google.com/; report-uri https://example.org"*

Sie sehen, dass **jede CSP-Variante** im Header auftaucht, damit sich möglichst viele Browser ansprechen lassen. Innerhalb der jeweiligen Header-Namen ist der Inhalt identisch: Quellen werden nacheinander aufgeführt und die Direktiven sind mit einem Semikolon voneinander getrennt. Die Syntax ist also immer die gleiche. Tatsächlich ändert sich nur der Name des Felds, weshalb Sie den Inhalt auch einfach duplizieren können.

Zunächst legen wir fest, dass – sollte es nicht anders in einer Direktive hinterlegt sein – Daten aus keiner Quelle geladen werden sollen (*default-src*). Damit schließen wir eine Sicherheitslücke. Sie sollten ***default-src* immer als erstes definieren**. So verhindern Sie, dass durch eine eventuell vergessene Direktive eine Lücke in Ihrer Content-Security-Policy verbleibt.

Als nächstes definieren wir die Quelle, aus denen Skripte geladen werden dürfen (*script-src*). Im Beispiel legen wir fest, dass der Browser nur Skripte von der gleichen Quelle und von *example.com* inklusive aller Subdomains lädt (die Wildcard vergeben Sie durch *\*.*). Des Weiteren bestimmen wir, dass Clients auch Stylesheets nur aus der eigenen Quelle laden dürfen (*style-src*). Auch Bilder sind nur aus der eigenen Quelle und als Data-URL erlaubt (*img-src*). Schriftarten dürfen gemäß unseres Content-Security-Policy-Headers nur noch vom eigenen Ursprung und aus dem Angebot von Google geladen werden. Schließlich geben wir im Beispiel noch einen Ort an, an den Meldungen geschickt werden, falls jemand versucht, gegen den Sicherheitsstandard zu verstoßen (*report-uri*).

Sie bemerken, dass wir bei weitem **nicht alle Direktiven in den Header** **aufgenommen** haben. Das ist auch gar kein Problem: Im angenommenen Fall brauchen wir keine weiteren Whitelists, und durch die *default-src* sind alle anderen Quellen ausgeschaltet.

[Tipp: Im Internet sind mehrere Content-Security-Policy-Generatoren verfügbar. Mit Angeboten wie CSP Is Awesome [Zum Generator von CSP Is Awesome] (http://cspisawesome.com/) oder Report URI [Zum Generator von Report URI] (https://report-uri.io/home/generate) erstellen Sie CSP-Header-Felder leicht und zuverlässig.]

**Teaser**

Websites können voller Sicherheitslücken sein, besonders wenn es um aktive Inhalte wie JavaScript geht. Durch Cross-Site-Scripting können Cyberkriminelle diesen Content für ihre Zwecke verändern. Das ist gefährlich für die Betreiber der Seite, aber vor allem für die Internetnutzer. Die Content-Security-Policy versucht dieses Risiko zu verringern. Erfahren Sie, was hinter dem Sicherheitsstandard steckt und wie dieser funktioniert.