Schutz vor Cross-Site-Scripting Angriffen mit Hilfe von Content Security Policy

Ausarbeitung für das Studienfach „Sichere Systeme“
an der Hochschule der Medien Stuttgart

von
Pascal Naujoks
(Matrikel-Nr.: 23397)

Stuttgart, September 2011

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Dieses Dokument befasst sich mit der Frage wie Cross-Site-Scripting (XSS) Angriffe auf Web-Anwendungen mit Hilfe einer Content Security Policy (CSP) vermieden werden können. Dabei setzt diese Arbeit voraus, dass die Problematik von XSS-Angriffen [1] und deren Funktionsweise [2] bereits bekannt ist und geht daher nicht näher auf diese Themen ein. Auch Begriffe wie reflektiertes XSS [3] und Clickjacking [4] sollten dem Leser bekannt sein.

Zunächst wird in Kapitel 3 auf den aktuellen Entwicklungsstand sowie die Daseinsberechtigung und generellen Funktionsweise einer CSP eingegangen. Da die Implementation von CSP bis Dato browserabhängig ist, wird desweiteren auf den unterschiedlichen Integrationsstand in den verschiedenen Browsern eingegangen. Kapitel 3.1 beschreibt dabei gesondert die bisherige Implementation in Microsofts Internet Explorer. Im Abschluss von Kapitel 3 werden die Grenzen einer CSP beim Schutz gegen XSS-Angriffe erläutert.

Im Rahmen dieser Arbeit wurde zusätzlich eine Extension für das Open Source Content Management System TYPO3 entwickelt, dass die einfache Integration einer CSP in die eigene Webseite ermöglicht. Eine Erläuterung dieser Extension und ein Fazit zu CSP bilden den Abschluss dieser Arbeit.

2 Cross-Site-Scripting (XSS)

Das Open Web Application Security Project sowie das Web Application Security Consortium führen Cross Site Scripting (XSS) nach wie vor unter den Top 3 der häufigsten Sicherheitslücken bei Web-Anwendungen auf. Dies zeigt die aktuelle Liste der „OWASP Top 10 Web Application Security Risks for 2010“ [5], sowie die folgende Auswertung der Web-Hacking-Incident-Database (WHID) [6] des Web Application Security Consortiums:

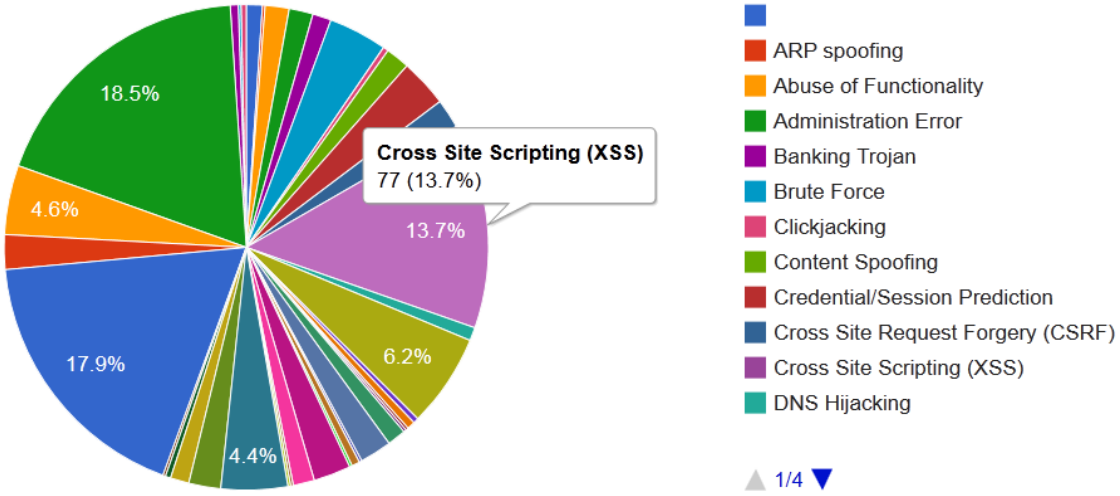


Abbildung 1: Häufige Angriffsmethoden auf Webanwendungen nach WHID[1] 2010

OWASP Top 10 – 2007 (Previous)		OWASP Top 10 – 2010 (New)
A2 – Injection Flaws	↑	A1 – Injection
A1 – Cross Site Scripting (XSS)	↓	A2 – Cross Site Scripting (XSS)
A7 – Broken Authentication and Session Management	↑	A3 – Broken Authentication and Session Management
A4 – Insecure Direct Object Reference	=	A4 – Insecure Direct Object References
A5 – Cross Site Request Forgery (CSRF)	=	A5 – Cross Site Request Forgery (CSRF)
<was T10 2004 A10 – Insecure Configuration Management>	+	A6 – Security Misconfiguration (NEW)
A10 – Failure to Restrict URL Access	↑	A7 – Failure to Restrict URL Access
<not in T10 2007>	+	A8 – Unvalidated Redirects and Forwards (NEW)
A8 – Insecure Cryptographic Storage	↓	A9 – Insecure Cryptographic Storage
A9 – Insecure Communications	↓	A10 – Insufficient Transport Layer Protection
A3 – Malicious File Execution	-	<dropped from T10 2010>
A6 – Information Leakage and Improper Error Handling	-	<dropped from T10 2010>

Abbildung 2: Häufige Angriffsmethoden auf Webanwendungen nach OWASP[2] 2010

[1] Abgerufen am 17.09.2011 von http://www.google.com/fusiontables/DataSource?snapid=S195929w1ly

Wie in Abbildung 2 zu erkennen ist, zählt XSS seit Jahren zu den Haupt Angriffsvektoren auf Webseiten. Dabei sollten die Basis-Techniken zur Verhinderung von XSS-Angriffen den meisten Webmastern inzwischen bekannt sein, gibt es hierzu doch ausreichend Literatur und Dokumente im Internet [7]. Das Problem könnte also „an der Wurzel“ beseitigt werden, würden die Webmaster ihre Webseiten entsprechend den Ratschlägen absichern. Leider wird dies jedoch in der Praxis nicht konsequent angewandt. Die Gründe hierzu können vielseitig sein: Mangelnde Implementation der Validatoren und Escaping, neue („zero-day“) Angriffstechniken, Fehlendes Bewusstsein des Risikos oder schlicht die Faulheit des Einzelnen sich immer wieder aufs Neue mit diesem lästigen Thema zu befassen. Hier kommt CSP ins Spiel. CSP ist Browserbasiert und muss nur einmalig vom Webmaster implementiert werden. Ist die Implementation gelungen und unterstützt der Browser des Besuchers CSP, muss sich der Webmaster theoretisch nicht mehr um die Absicherung seiner Skripte gegen XSS kümmern.

Leider befindet sich jedoch die Entwicklung einer Norm für die Einführung von CSP beim W3C Consortium nach wie vor in der „Unofficial Draft“ Version [8]. Demnach gibt es noch keine finale normierte Richtlinie, wie CSP von Webmastern und den Browserherstellern zu implementieren ist. Hauptsächlich wird das Dokument von der Brandon Sterne, dem Security Program Manager bei der Mozilla Foundation aktualisiert. Dies hat zur Folge, dass der Firefox Browser (ab Version 4) am ehesten den Spezifikation entspricht und vice versa. Neben dem Mozilla Firefox haben zum Zeitpunkt der Erstellung dieses Dokuments nur wenige andere Browser CSP implementiert. Microsoft verfolgt zwar schon länger einen anderen Weg Content Security durchzusetzen (Siehe Kapitel 3.1: Schutz vor XSS im Microsofts Internet Explorer), dies wird jedoch nur von den wenigsten Webentwicklern implementiert. Mit dem vorgestellten Draft des W3C wächst die Hoffnung auf eine einheitliche, rückwärtskompatible und einfach zu implementierende Content Security Policy zum Schutz vor XSS-Angriffen.

[2] Abgerufen am 17.09.2011 von http://owasptop10.googlecode.com/files/OWASP_Top_10_-_2010%20Presentation.pptx

3 Content Security Policy (CSP)

Das Grundproblem von XSS-Attacken ist, dass ein Webbrowser nicht weiß welcher Code einer Seite als vertrauenswürdig und welcher als bösartig zu interpretieren ist. Ein Webbrowser behandelt deshalb jedes Stück Code einer Webseite als Vertrauenswürdig und führt es aus. CSP gibt den Webmaster im Whitelist-Prinzip [9] die Möglichkeit zu definieren welche Art von Code und von welchen Quellen Code als Vertrauenswürdig einzustufen ist. Die CSP Direktiven werden in Form eines HTTP-Headers vom Webserver an den Browser gesendet. Dies stellt sicher, dass die Direktive nicht bereits in Folge eines XSS-Angriffs manipuliert wurde (Ausnahmen siehe Kapitel 3.4, Grenzen von CSP).

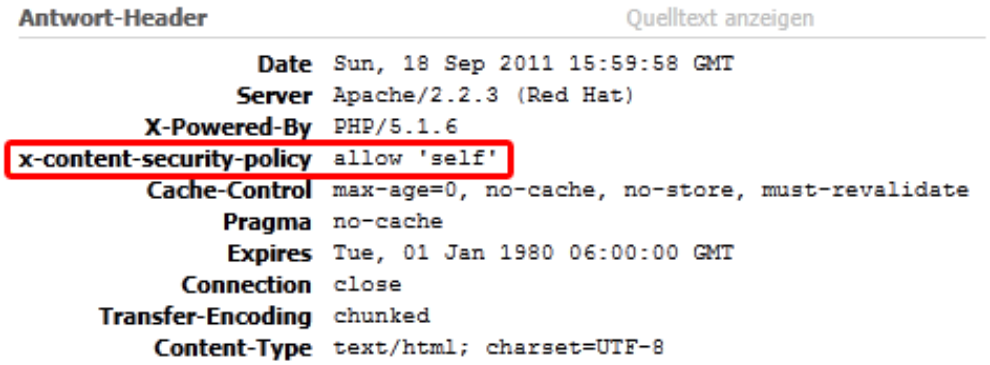


Abbildung 3: Eine gesetzte CSP Direktive im HTTP-Header einer Webseite

Unterstützt der Browser des Besuchers CSP und hat der Webmaster CSP nach den Vorgaben des Browserherstellers implementiert, so wird jeder Code, der nicht in der Whitelist enthalten ist, als bösartig eingestuft und nicht geladen oder ausgeführt. Optional wird jede Verletzung der CSP an eine gesonderte URL gesendet, so dass der Webmaster Sicherheitslücken in seiner Anwendung bereits im Kern beheben kann.

Ist CSP nicht auf einer Webseite aktiv oder wird dies von einem Webbrowser nicht erkannt, so fällt der Browser in den schon länger bekannten Same-Origin Modus[3] zurück. Die dort greifende Same-Origin-Policy verhindert lediglich, dass der Inhalt einer Webseite A den Inhalt einer Webseite B verändert. Same-Origin bedeutet in diesem Falle gleiche Domain, Protokoll und Port.

Doch wer unterstützt CSP? Wie in Kapitel 1 beschrieben, ist CSP aktuell noch ein Unofficial Draft und daher für viele Browserhersteller noch nicht attraktiv genug, um es

[3] Details siehe: http://de.wikipedia.org/wiki/Same-Origin-Policy

in den Kern ihrer Produkte zu implementieren. Neben dem bereits erwähnten Firefox 4 (und neuer) unterstützt aktuell nur Google Chrome ab Version 13[4] CSP.

Abbildung 4: Browser, markiert nach ihrer Unterstützung für CSP

Im Opera Browser wird eine Erweiterung benötigt und im Safari Browser ist die Implementation von CSP nur in Planung. Microsofts Internet Explorer ab Version 8 hat bereits einen kleinen XSS-Filter[5], mit dem Typ1-XSS Angriffe (Siehe Kapitel 3.1, Schutz vor XSS im Microsofts Internet Explorer) abgefangen werden sollen. Dies soll geschehen, ohne dass eine gesetzte CSP nötig ist. Alle anderen Browser Hersteller haben sich noch nicht zum Thema CSP geäußert.

3.1 Schutz vor XSS im Microsofts Internet Explorer

Microsoft hat bereits 1999 vor der Veröffentlichung von CSP in seinem Internet Explorer 8 erste Maßnahmen gegen XSS eingeführt. Der von Microsoft als „Siteübergreifender Skriptfilter“ bezeichnete Schutzmechanismus kann jedoch ausschließlich „reflektiertes (Typ 1) XSS“ abfangen. Bei einem erkannten Typ 1 XSS-Angriff unterbindet der Internet Explorer die Kommunikation mit dem Zielserver des Skripts und gibt dem Benutzer eine entsprechende Warnmeldung aus. Dieses Verhalten ist standardmäßig im Internet Explorer aktiv, ohne dass der Mechanismus explizit durch einen Webmaster aktiviert werden muss. Nutzt eine Webseite weitere Server zur Kommunikation mit dem Besucher so muss der Webmaster durch einen gesondert gesendeten HTTP-Header den Siteübergreifender Skriptfilter abschalten. Die Angabe einer Whitelist ist nicht möglich.

Eine weitere Maßnahme soll dem Benutzer des Internet Explorers ab Version 8 vor Clickjacking schützen. Der von Microsoft als „X-Frame-Options“ (XFO) betitelten HTTP-Header kann von einem Webmaster mit zwei verschiedenen Werten versehen werden:

1. deny: Ist dieser Wert gesetzt verhindert der Internet Explorer, dass der Inhalt der Ursprungsseite in einem Frame oder iFrame dargestellt wird.

[4] Quelle: http://blog.chromium.org/2011/06/new-chromium-security-features-june.html

[5] Quelle: http://blogs.technet.com/b/srd/archive/2008/08/19/ie-8-xss-filter-architecture-implementation.aspx

2. SameOrigin: Ist dieser Wert gesetzt kann die Webseite ausschließlich von einem Frame oder iFrame geladen werden, der dieselbe Domain aufweist wie die Ursprungsseite.

Im Gegensatz zum Siteübergreifenden Skriptfilter muss dieser HTTP-Header explizit gesetzt werden.

Da beide Maßnahmen ebenfalls auf das Senden eines HTTP-Headers aufbauen sind, wie in Kapitel 3.4 - Grenzen von CSP erwähnt, auch diese Schutzmechanismen durch einen Angreifer manipulierbar und somit kein vollständiger Schutz vor XSS-Angriffen.

3.2 CSP Basis Restriktionen

Unterstützt und erkennt ein Browser eine CSP Direktive, so gelten fünf weitere Basis Restriktionen. Diese Restriktionen unterbinden die bereits im Web als „*the bad parts of JavaScript*“ bekannten Paradigmen, wie JavaScript nicht zu verwenden ist. Im Buch „*JavaScript: The Good Parts*“ [10] wird näher auf diese eingegangen. Im Folgenden werden die fünf Basis Restriktionen von CSP erläutert, die einen Teil der „*bad parts*“ unterbinden.

1. **Inline JavaScript und CSS wird unterbunden:** JavaScript, das inline oder über Attribute geladen wird, wird nicht mehr ausgeführt. Desweiteren können JavaScript Funktionen nicht mehr über Inline function calls aufgerufen werden.

   Dies verhindert die Ausführung von *Reflected XSS, Stored XSS, Link Injection* sowie *HTML Attribute Injection.* Diese Restriktion bringt den größten Nutzen mit sich, wird aber für viele Webseiten Betreiber auch die größte Hürde beim Einsatz von CSP darstellen. Während das Verschieben von Inline JavaScript in externe Dateien kein großes Problem darstellt, stellt das Umstellen von *Inline Function Calls* auf das *Event Listener* Prinzip bei den meisten Webprojekten einen größeren Kraftakt dar.

2. **Code aus Strings wird unterbunden:** Code, der Initial als String vorhanden ist und erst zur Laufzeit in Code umgewandelt wird, wird nicht mehr ausgeführt.

   Damit ist es unter anderem nicht mehr möglich die zu den „bad parts“ gehörenden JavaScript Funktionen eval() zu benutzen und anonymen Code in Form von Strings zu erzeugen.

3. ***data* ist keine vertrauenswürdige Quelle mehr.** Objekte deren Inhalt aus (codierten) *data* Strings besteht, werden nicht mehr geladen.

   Mit dem *data* Schema ist es möglich Text, Bilder und weitere Objekte (codiert) als Zeichenkette abzuspeichern und anderen Objekten als Quelle zur Verfügung zu stellen. Diese Technik wird nur selten genutzt und sollte ebenfalls kein Hindernis bei der Migration von CSP darstellen. Diese Restriktion verhindern die häufig in Forenlinks und Phising E-Mails verwendete *URL Script Injection.*

4. **XBL bindings dürfen nur noch von *chrome:* und *resource:* Protokollen geladen werden**

   Die XBL Binding Sprache ist eine proprietäre Entwicklung von Mozilla, mit der man das Verhalten und Aussehen von Webseiten beschreiben kann. Sie wird aktuell nur von Browsern, die die Gecko-Engine nutzen unterstützt und findet aufgrund dieser Tatsache im Web nur wenig bis gar keine Verwendung.

5. **Policy- und Report-URLs dürfen nur zum selben Host verweisen**. Dies betrifft die von CSP eingeführte *policy-uri* sowie die *report-uri.*

   Sämtliche CSP-Direktiven können in einer separaten Datei abgelegt werden. Diese Datei, die durch die *policy-uri* angegeben wird, muss sich jedoch nach der fünften Basisrestriktion auf demselben Host befinden. Das gleiche gilt für die *report-uri,* an die Verletzungen der gesetzten CSP gesendet werden können. So ist es mit den Auswertungen, die an die *report-uri* gesendet werden, möglich Schwachstellen im eigenen System aufzudecken und zu beheben.

3.3 CSP Direktiven und Scopes

CSP Direktiven werden in Form eines HTTP-Header an den Webbrowser gesendet. Dieser HTTP-Header wird bezeichnet durch den String *„X-Content-Security-Policy“* und enthält als Wert bis zu 13 Policies. Eine Policy beschreibt von welchen Quellen Inhalt (*Content Sources*) geladen werden darf. Gesetzt werden dürfen drei verschiedene Arten von *Content Sources:*

1. **Host-Angabe:** Angabe eines Hosts in Form von http://domain.tld, ftp://domain.tld, usw. Das Setzen von Wildcards ist ausschließlich in der Form *.domain.tld möglich. Das Setzen von Wildcards innerhalb oder nach dem Hostnamen ist nicht möglich. Optional kann noch ein Port gesetzt werden (domain.tld:443). Die Angabe eines * ohne die Angabe eines Hosts deaktiviert die jeweilige CSP Direktive und Inhalt darf wieder von jeder Quelle geladen werden.
2. **Keyword-Angabe:** Verwendung einer der beiden Keywords *'self'* oder *'none'*. *'self'* beschreibt die eigene Domain unter der die Webseite geladen wurde. Dies ist Hilfreich wenn die Webseite unter mehreren Adressen erreichbar ist. *'none'* besagt, dass kein Host erlaubt ist und keine weiteren Inhalte geladen werden dürfen. Beide Keywords müssen in einfachen Anführungszeichen angegeben werden.
3. **data-Angabe:** Verwendung des Keywords *data. data* ermöglicht das Laden von Inhalt unabhängig von der Angabe eines Hosts mithilfe einer data URL.

Diese Content Sources können bei den einzelnen Policies als Wert gesetzt werden. Die Policies sind aufgeteilt in eine *Basis Policy*, eine *Option Policy*, neun *Scope Policies*, die *Report Policy* sowie die *Policy URL*.

Die erste Policy „*allow*“ beschreibt das Standardverhalten der Policies 3 bis 11 und ist zugleich Pflicht beim Einsatz von CSP. Die hier gesetzte Content Source gilt für alle Scope Policies, sofern diese nicht explizit durch eine solche überschrieben werden.

Die zweite Policy kann die in Kapitel 3.2 beschriebenen Basis Restriktionen zum Teil überschreiben. Dies betrifft das reaktivieren von Inline-Script, der eval()-Funktion und das generieren von Code aus Strings.

Die Policies 3 bis 11 sind die Scope Policies und beschreiben die erlaubten Content Sources für Bilder (*img-src*), Audio- und Video-Tags (*media-src*), JavaScript (*script-src*), eingebettete Skripte und Applets (*object-src*), Frames und Iframes (*frame-src*), Schriften (*font-src*), Stylesheets (*style-src*) sowie das Verhalten der Webseite innerhalb eines Frames einer anderen Webseite (*frame-ancestor*).

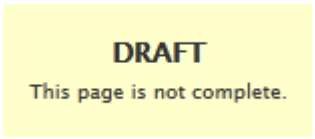
Die vorletzte Policy *report-uri* gibt an, an welche Adresse eine Verletzung der CSP gesendet wird. Die übertragenen Daten beinhalten die Request-Methode, den Request-Header, die geblockte URL, die verletzte Policy sowie die komplette CSP der Webseite. Die Daten werten im JavaScript Object Notation[6] (JSON) übermittelt und müssen dementsprechend vom Zielskript verarbeitet werden.

Die letzte Policy gibt dem Webmaster die Möglichkeit die komplette CSP in einer Datei abzulegen. Die *policy-uri* akzeptiert jedoch, wie auch die *report-uri,* aus Sicherheitsgründen lediglich den eigenen Host als Content Source.

Ein CSP kann nach den genannten Regeln folgendermaßen aussehen:

```
X-Content-Security-Policy: allow 'self'; img-src *; javascript-src google.com;
```

Dieses Beispiel erlaubt Standardmäßig als Content Source jeglichen Inhalt des eigenen Hosts (`allow 'self';`). Diese Regel wird für Bilder im folgenden Befehl überschrieben durch `img-src *;`. Diese besagt, dass Bilder von jeglichen Quellen geladen werden dürfen. Das Laden von JavaScript ist jedoch ausschließlich von google.com erlaubt (`javascript-src google.com;`).



Die detaillierte Spezifikation, wie sie im Firefox Browser implementiert ist und als Grundlage für dieses Kapitel diente, ist unter [11] zu finden. Jedoch ist diese ebenfalls noch in der Draft-Version und mit Vorsicht zu genießen.

[6] Erläuterung von JSON siehe http://de.wikipedia.org/wiki/JSON

3.4 Grenzen von CSP

Geht man davon aus, dass man einen Browser verwendet der CSP unterstützt und dass der Webseiten Betreiber die nötigen CSP Direktiven gesetzt hat, so hat man auch dann keinen vollständigen Schutz vor XSS. Bis Dato sind zumindest zwei Möglichkeiten bekannt, mit denen man trotz einer aktiven CSP erfolgreich XSS-Angriffe durchführen kann.

Die erste Möglichkeit besteht darin, eine Webseite oder Webserver so zu manipulieren, dass Schadhafte Skripte auf dem Webserver abgelegt und regulär geladen werden. In diesem Fall spricht man von *konsistentem XSS*. Auf diese Art wird das Whitelist Prinzip von CSP ausgenutzt, um die schadhaften Skripte wieder als Vertrauenswürdig einstufen zu lassen. Voraussetzung für diese Art von Angriff ist eine Sicherheitslücke im Content Management System der Webseite oder dem Webserver.

Ein weiteres Manko in der aktuellen Implementation von CSP ist die Übermittlung der Whitelist an den Browser. Dies erfolgt, wie in Kapitel 3 erläutert, über einen zusätzlichen HTTP-Header. Diese Übergabe kann von einem Angreifer über eine Header Injection [12] abgefangen und gelöscht oder mit einer eigenen Whitelist überschrieben werden.

Nimmt man diese beiden Möglichkeiten CSP zu umgehen näher in Betracht, so stellt man fest, dass CSP keine endgültige Lösung sondern lediglich eine Verschiebung des Problems darstellt.

4 Implementation einer CSP am Beispiel von TYPO3

Im Rahmen dieser Arbeit wurde eine Extension entwickelt, die die Integration einer CSP in das Open Source Content Management System TYPO3 erleichtern soll. Die Extension ist im öffentlichen TYPO3 Extension Repository hinterlegt und unter folgender Adresse zu finden:

http://typo3.org/extensions/repository/view/itspn_csp/current/

Nach der Installation steht ein neues Backend Modul „Content-Security-Policy“ zur Verfügung (siehe Abbildung 5Abbildung 1), das dem Benutzer die Möglichkeit gibt seine CSP auf zwei verschiedene Arten anzulegen: Mit Hilfe eines „Wizards“ oder manuell. Im Folgenden werden diese beiden Möglichkeiten näher beschrieben.

Abbildung 5: Das Backend der TYPO3 CSP Extension

4.1 Anlegen einer CSP mit Hilfe des Wizards

Die erste Möglichkeit zum Erzeugen einer CSP ist das Ausführen des „Content-Security-Policy Recommendation Wizard“ (siehe Abbildung 6). Dieser Wizard ermittelt mit Hilfe eines „CSP Recommendation Bookmarklets“ von Brandon Sterne[7] eine CSP Empfehlung für die eigene Webseite. Das CSP Recommendation Bookmarklet ist ein JavaScript, das den Quellcode der Webseite durchsucht, nach allen gesetzten Quellen aller möglichen CSP Direktiven sucht und diese aufzeichnet. Beispielsweise wird abgefragt, von welchen Quellen JavaScript-Dateien geladen werden. Alle gefundenen Quellen für JavaScript werden dann in die Empfehlung für die CSP Direktive *script-src* eingesetzt. Dabei geht das CSP Recommendation Bookmarklet davon aus, dass die

[7] CSP Bookmarklet: http://brandon.sternefamily.net/posts/2011/01/update-to-csp-bookmarklet/

Webseite aktuell kein Opfer eines XSS-Angriffes ist. Der Wizard, der sich dieses Skript zu Nutze macht, läuft in drei Schritten ab:

1. Eingabe der URL für die eine CSP Empfehlung erstellt werden soll. Diese ist in den meisten Fällen die Startseite der eigenen Webseite. Nach dem Absenden des Formulars wird die eingegebene URL in einem iFrame geladen. So ist es möglich, die Ausgabe des Frontends der Webseite zu erhalten und das CSP Recommendation Bookmarklet basierend auf dieser Ausgabe laufen zu lassen.
2. Die Empfehlungen des CSP Recommendation Bookmarklets werden in ein Formular geladen, das alle aktuell zur Verfügung stehenden CSP Direktiven aufführt. Dieser Vorgang ist in der folgenden Abbildung dargestellt:

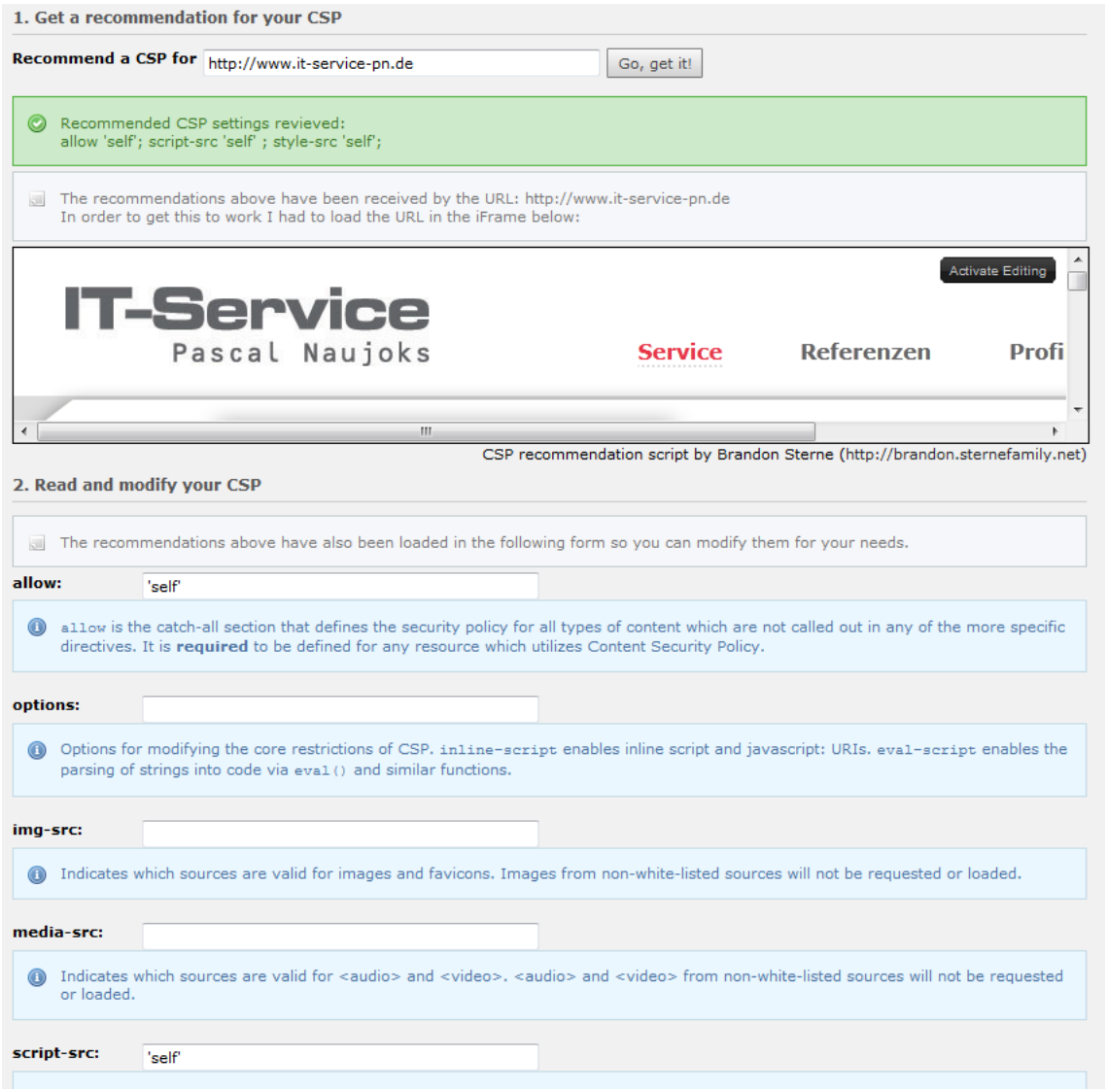


Abbildung 6: Der „CSP Recommendation Wizard“

3. Der Administrator der Webseite hat nun die Möglichkeit die Eingaben zu modifizieren und abzuspeichern. Dies ist z.B. nötig wenn auf Unterseiten andere CSP Direktiven gelten, als auf der Startseite.

4.2 Manuelle Modifikation der CSP

Die zweite Möglichkeit eine CSP anzulegen ist das direkte Editieren der CSP Direktiven. Hierzu muss der Administrator der Webseite auf das zweite Modul „Your current Content-Security-Policy“ wechseln (siehe Abbildung 7). In diesem Modul sind ebenfalls alle CSP Direktiven aufgeführt und können direkt editiert werden:

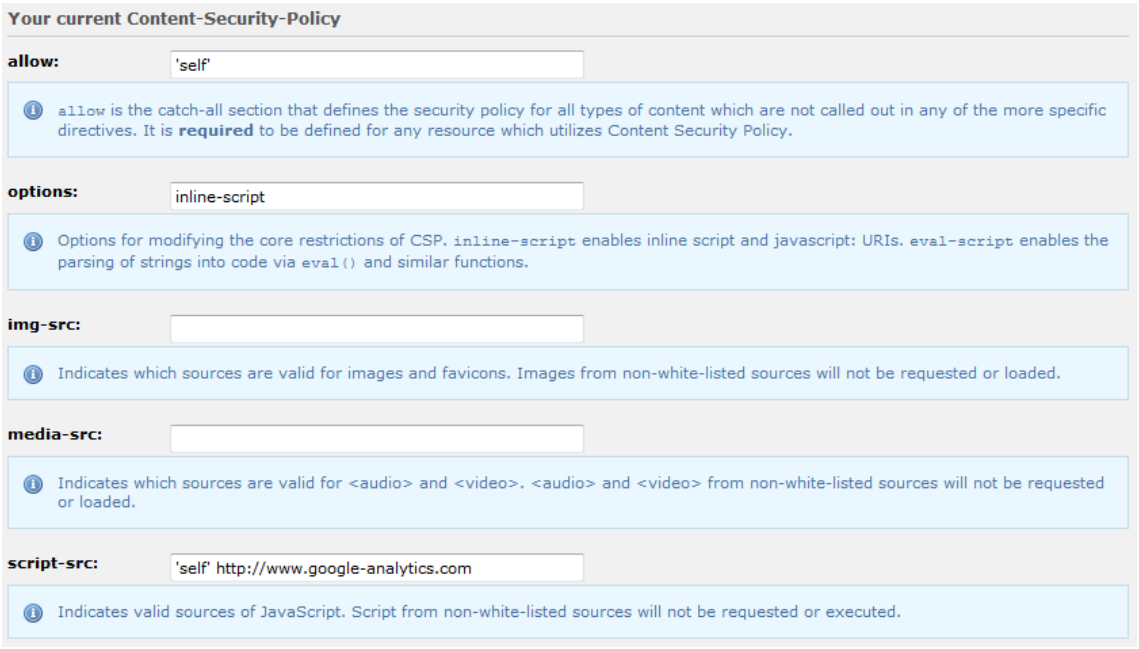


Abbildung 7: Manuelle Modifikation der CSP Direktiven

4.3 Aktivierung der CSP

Unabhängig von der gewählten Art der Erzeugung einer CSP muss diese zunächst aktiviert werden. Dies erfolgt über den eingebauten Konstanten-Editor von TYPO3 (siehe Abbildung 8). Durch einen Klick auf die Checkbox ist die aktuell gespeicherte CSP im Frontend der Webseite aktiv.

Abbildung 8: Aktivierung der CSP mit Hilfe des Konstanten-Editors von TYPO3

Nach erfolgter Aktivierung greifen alle gesetzten CSP Direktiven und werden fortan als HTTP-Header an den Browser gesendet. Dies geschieht im Wesentlichen in folgenden sieben Zeilen Code:

```
$res = $GLOBALS['TYPO3_DB']->exec_SELECTquery('*', 'tx_itspncsp_directives', '', '', '', '1');
$row = $GLOBALS['TYPO3_DB']->sql_fetch_assoc($res);

if($res && is_array($row) && sizeof($row['directives']) && $conf['csp.']['activate']){
    $GLOBALS['TSFE']->config['config']['additionalHeaders'] = 'X-Content-Security-Policy: '.$row['directives'].
            '|'.$GLOBALS['TSFE']->config['config']['additionalHeaders'];
}
```

Abbildung 9: Senden der CSP an den Browser

In den ersten beiden Zeilen werden alle gespeicherten CSP Direktiven aus der Datenbank mit Hilfe der von TYPO3 zur Verfügung gestellten Datenbank Wrapper Klassen ausgelesen. In Zeile vier wird geprüft ob die Datenbank-Abfrage erfolgreich war und die CSP Extension aktiviert wurde. In den Zeilen fünf und sechs wird die ausgelesene CSP an die bisher von TYPO3 deklarierten HTTP-Header angehängt.

4.4 Die Reporting Funktion einer CSP

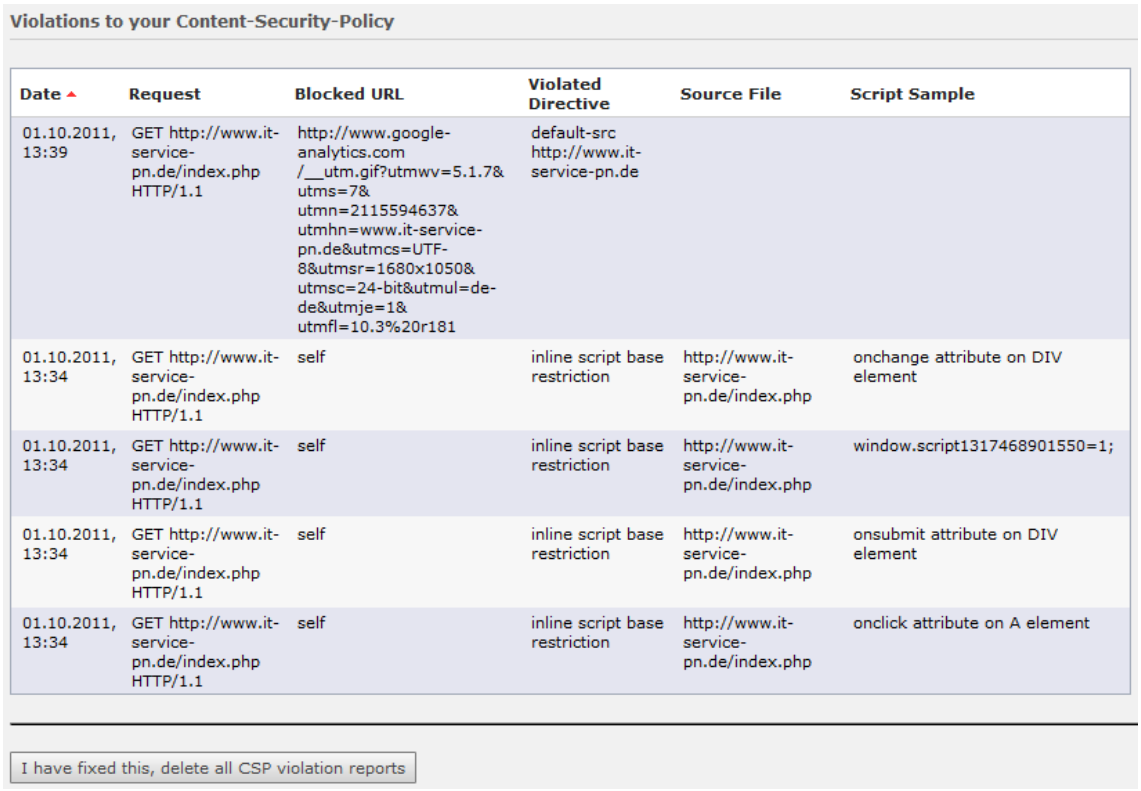
Bei der Verletzung einer CSP Direktive wird diese vom Browser an die in der CSP Direktive „*report-uri*“ angegeben URL weitergeleitet. Bei der CSP Extension für TYPO3 ist dies die aktuell vom Besucher aufgerufene URL. Da die Extension auf jeder Seite eingebunden ist, kann sie somit die vom Browser zur Verfügung gestellten Informationen jederzeit entgegen nehmen. Die im JSON-Format zur Verfügung gestellten Informationen werden geparst und als Dump abgespeichert:

```
$json = file_get_contents('php://input');
$violations = json_decode($json, true);
$violation = $violations['csp-report'];

if ($json !== false && is_array($violation) && sizeof($violation)) {

    $dump = print_r($violation,true);
    $violRes = $GLOBALS['TYPO3_DB']->exec_SELECTquery('uid',
            'tx_itspncsp_violations','dump="'.$dump.'"', '', '', '1');

    if(!$GLOBALS["TYPO3_DB"]->sql_num_rows($violRes)){
        $data = array(
            'crdate' => time(),
            'tstamp' => time(),
            'request' => $violation['request'],
            'requestheaders' => $violation['request-headers'],
            'blockeduri' => $violation['blocked-uri'],
            'violateddirective' => $violation['violated-directive'],
            'originalpolicy' => $violation['original-policy'],
            'sourcefile' => $violation['source-file'],
            'scriptsample' => $violation['script-sample'],
            'linenumber' => $violation['line-number'],
            'dump' => $dump
        );

        $GLOBALS['TYPO3_DB']->exec_INSERTquery('tx_itspncsp_violations', $data);
    }
}
```

Abbildung 10: Abspeichern einer Verletzung einer CSP Direktive

In den ersten drei Zeilen in Abbildung 10 wird der übermittelte Report im JSON-Format von der Standardeingabe („php://input“) entgegen genommen. Dieser wird dann von einer PHP-Funktion in ein Array umgewandelt („json_decode“). Aus diesem Array wird mit Hilfe von „print_r“ ein Fingerprint der CSP Verletzung erzeugt und mit den bisher gemeldeten Einträgen in der Datenbank verglichen (Zeile 8). Gibt es noch keinen Eintrag in der Datenbank, der dieser CSP Verletzung gleicht (Zeile 10), so wird von der Extension ein neuer Eintrag mit sämtlichen zur Verfügung stehenden Daten angelegt (Zeile 11 bis 25).

Durch die Speicherung des zur Verfügung gestellten Reports ist es möglich, die Verletzungen tabellarisch aufzubereiten und im Backend der Extension darzustellen:



Violations to your Content-Security-Policy

Date ▲	Request	Blocked URL	Violated Directive	Source File	Script Sample
01.10.2011, 13:39	GET http://www.it-service-pn.de/index.php HTTP/1.1	http://www.google-analytics.com /__utm.gif?utmwv=5.1.7&utms=7&utmn=2115594637&utmhn=www.it-service-pn.de&utmcs=UTF-8&utmsr=1680x1050&utmsc=24-bit&utmul=de-de&utmje=1&utmfl=10.3%20r181	default-src http://www.it-service-pn.de		
01.10.2011, 13:34	GET http://www.it-service-pn.de/index.php HTTP/1.1	self	inline script base restriction	http://www.it-service-pn.de/index.php	onchange attribute on DIV element
01.10.2011, 13:34	GET http://www.it-service-pn.de/index.php HTTP/1.1	self	inline script base restriction	http://www.it-service-pn.de/index.php	window.script1317468901550=1;
01.10.2011, 13:34	GET http://www.it-service-pn.de/index.php HTTP/1.1	self	inline script base restriction	http://www.it-service-pn.de/index.php	onsubmit attribute on DIV element
01.10.2011, 13:34	GET http://www.it-service-pn.de/index.php HTTP/1.1	self	inline script base restriction	http://www.it-service-pn.de/index.php	onclick attribute on A element

I have fixed this, delete all CSP violation reports

Abbildung 11: Ausgabe aller Verletzungen der gesetzten CSP

Mit diesen Daten ist es dem Administrator der Webseite möglich, die gesetzten CSP Direktiven weiter zu verbessern oder Sicherheitslücken in seinen Skripten ausfindig zu machen.

5 Fazit

Sicherheit wird in der Praxis häufig durch verschiedene Security-Layer abgebildet. CSP ist nur ein weiterer Spielzug im Ping-Pong Spiel zwischen den Hackern auf der einen und den Webmastern auf der anderen Seite. Bei einer konsequenten Implementation aller Browserhersteller hat eine CSP als zusätzlicher Security-Layer durchaus seine Daseinsberechtigung. Kapitel 3.4 „Grenzen von CSP“ hat jedoch gezeigt, dass selbst dann kein vollständiger Schutz vor XSS-Angriffen besteht, wenn Webseiten Betreiber und Webbrowser CSP oder XFO unterstützen.

Eine endgültige Lösung des Problems kann nur darin bestehen, dass alle Webmaster Ihre Anwendungen bereits im Kern vor Angriffen absichern und Browserhersteller ihre Benutzer unabhängig davon zusätzlich vor XSS-Angriffen schützen. In der Praxis ist man jedoch noch weit davon entfernt. Wenn man davon ausgeht, dass eine Webseite stand heute sicher ist vor XSS-Angriffen, so kann diese bereits morgen gegen eine neue XSS-Attacke verwundbar sein. In diesem Falle könnte CSP einen Großteil aller Webseiten und Besucher vorerst schützen. Denn mit jedem Schlagabtausch steigt die Komplexität des Angriffes und somit sinkt auch die Zahl derer, die den Angriff durchführen können.

6 Literaturverzeichnis

[1] Einen Einstieg in das Thema XSS bietet Wikipedia unter http://de.wikipedia.org/wiki/Cross-Site-Scripting sowie das OWASP Project unter https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29 (Abgerufen am 17.09.2011).

[2] Die gängigsten XSS-Angriffsvektoren und deren „Browserkompatibilität“ stellt Robert "RSnake" Hansen unter http://ha.ckers.org/xss.html zur Verfügung. Abgerufen am 17.09.2011

[3] XSS-Angriffe erklärt und in unterschiedliche Typen und Klassen unterteilt: http://www.xssed.com/xssinfo#Type_1 (Abgerufen am 18.09.2011).

[4] Clickjacking Erklärung auf Wikipedia: http://de.wikipedia.org/wiki/Clickjacking (Abgerufen am 27.09.2011)

[5] OWASP Top 10 - 2010 Document: http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202010.pdf (Abgerufen am 17.09.2011).

[6] Web-Hacking-Incident-Database: Die WHID bietet die Möglichkeit Angriffe auf Webseiten statistisch zu erfassen und mit Hilfe von Google Fusion Tables auszuwerten. http://projects.webappsec.org/w/page/13246995/Web-Hacking-Incident-Database (Abgerufen am 17.09.2011).

[7] „XSS Attacks - Cross Site Scripting Exploits and Defense“ von Seth Fogie (Autor), Jeremiah Grossman, Robert Hansen, Anton Rager, und Petko Petkov. ISBN 1597491543. Sowie das XSS Prevention Cheat Sheet unter https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet (Abgerufen am 17.09.2011).

[8] Content Security Policy Draft des W3C Consortiums: https://dvcs.w3.org/hg/content-security-policy/raw-file/tip/csp-unofficial-draft-20110315.html (Abgerufen am 17.09.2011 als Unofficial Draft Version).

[9] Erläuterung der Funktionsweise von CSP von Brandon Sterne (Security Program Manager bei der Mozilla Foundation): http://people.mozilla.org/~bsterne/content-security-policy/details.html (Abgerufen am 17.09.2011).

[10] „JavaScript: The Good Parts“ von Douglass Crockford. Mai 2008. ISBN 978-0-596-51774-8. Siehe http://shop.oreilly.com/product/9780596517748.do

[11] CSP policy directives von der Mozilla Foundation: https://developer.mozilla.org/en/Security/CSP/CSP_policy_directives (Abgerufen am 18.09.2011 als Draft-Version)

[12] Whitepaper von Amit Klein (AKsecurity), „HTTP Response Smuggling“: http://lists.grok.org.uk/pipermail/full-disclosure/2006-February/042358.html (Abgerufen am 26.09.2011).